MONOGRAFÍA, ANALISIS COMPARATIVO DE LAS PRINCIPALES TÉCNICAS DE COMPRESIÓN DE DATOS

DANIEL STEVEN GIL CRUZ

UNIVERSIDAD TECNOLÓGICA DE PEREIRA
FACULTAD DE INGENIERIAS
PROGRAMA INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
PEREIRA
2018

MONOGRAFÍA, ANALISIS COMPARATIVO DE LAS PRINCIPALES TÉCNICAS DE COMPRESIÓN DE DATOS

DANIEL STEVEN GIL CRUZ

MONOGRAFÍA

INGENIERO OMAR IVAN TREJOS BURITICA

UNIVERSIDAD TECNOLÓGICA DE PEREIRA
FACULTAD DE INGENIERIAS
PROGRAMA INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
PEREIRA
2018

AGRADECIMIENTOS

Quiero agradecer a las personas que de una u otra forma participaron en la realización de este proyecto ya que por mínimo que fuese su aporte fue muy importante para la realización del mismo, por esto quiero decirles en general ¡Gracias! ya que participaron e hicieron posible la elaboración de esta monografía.

Contenido

INTRODUCCIÓN

En nuestra vida cotidiana muchas veces tenemos que manejar información por lo que pretendemos llevarlas de manera digital para una mayor facilidad al momento de utilizarlas, algunos ejemplos son: fotografías, videos, trabajos en diferentes formatos o la utilización de plataformas virtual como la nube.

En todo los casos en cuanto a medios de almacenamientos de información se tienen una capacidad, un límite, donde se requieren llevar a cabo el manejo de bits, por medio de los cuales se transmiten datos por los medios de comunicaciones existentes, estas razones nos llevan a usar los distintos tipos de técnicas de compresión de datos, donde se pretende aprovechar al máximo estos medios de almacenamiento de información, pero acceder a grandes capacidades de almacenamiento es bastante costoso, hoy por hoy podremos dimensionar las grandes cantidades de información que manejan las diferentes compañías, donde deberán transmitirlas y almacenarlas.

Por otra parte también podemos ver los diferentes dispositivos y medios de red, lo cuales han evolucionado bastante, aumentando su capacidad de transmisión de datos, fiabilidad, entre otros factores, pero para acceder a estas buena tecnologías debemos tener buen capital ya que muchos de estos dispositivos son bastante costos y es allí donde la compresión de datos se hace importante porque en el mismo espacio de almacenamiento, por medio de las diferentes técnicas de compresión de datos podremos ampliar la capacidad de estos dispositivos de almacenamiento.

Mediante un conocimiento claro de las técnicas modernas de compresión de datos se abren nuevos caminos para tener una teoría que permita posteriormente optimizar el uso de los medios de transmisión y el almacenamiento, haciendo posible el manejo de altos volúmenes con dispositivos no especializados y por lo tanto menos costosos.

El objetivo principal de esta monografía es analizar los fundamentos matemáticos de las principales técnicas de compresión de datos, para posteriormente establecer criterios comparativos entre estas técnicas, donde se hace especial énfasis en los medios de compresión sin pérdida y por último se hace una introducción a las técnicas de compresión con perdida.

Para realizar esta monografía se hizo un uso intensivo de la Internet en el levantamiento de la información correspondiente al estado del arte acerca de la teoría de la información, la compresión y el almacenamiento. Una vez obtenida una robusta y abundante base de información bibliográfica se establecieron generalidades en cuanto a la teoría recogida y evaluada; Ya con los fundamentos teóricos y matemáticos establecidos se pasó a una fase de estudio detallado de las diferentes técnicas de compresión de datos, posteriormente se hace un análisis completo de estas técnicas para llevar a cabo una comparativa entre ellas.

Como monografía se busca poder ampliar más la información acerca de la compresión de datos, para en un futuro utilizar mejor los diferentes dispositivos, donde se manipulen todo tipo de información ya que hasta el momento hemos sido simples usuarios de estos conceptos a nivel académico, se trata pues de colocar a nivel de la comunidad académica la información correspondiente a la teoría de la compresión, sus efectos, ventajas, desventajas y desglose matemático para que posteriormente se pueda pensar en nuevas formas de investigación y de aplicar esta teoría e incluso de innovarla.

1 TEORÍA DE LA INFORMACIÓN

1.1 INTRODUCCIÓN

Dentro del tema de la teoría de información, se deben tener en cuenta varios aspectos:

- Fuente: componente de naturaleza humana o mecánica que determina el tipo de mensaje que se transmitirá y su grado de complejidad.
- Codificador: Recurso técnico que transforma el mensaje originado por la fuente de información en señales apropiadas según el canal a utilizar.

- Canal: Medio generalmente físico que transporta las señales en el espacio (cumple funciones de mediación y transporte).
- Decodificador: Recurso técnico que transforma las señales recibidas a una forma entendible por el Destino.
- Destino: componente terminal del proceso de comunicación, al cual está dirigido el mensaje.

En este capítulo se estudiaran diferentes términos, como los tipos de fuentes existentes y algunas características de estos, donde hablaremos de tres componentes principales, que son importantes a la hora de la manipulación de datos para el flujo de información desde un emisor a un receptor.
El objetivo de este capítulo es cambiar la percepción sobre lo que se entiende por información, pasando de una visión cualitativa e intuitiva a una visión cuantitativa y analítica de la misma, luego de exponer los diferentes procesos matemáticos, se hace un análisis entre los diferentes algoritmos para posteriormente dar una visión más clara, de cual podría surtir nuestra necesidad.

Para llegar a una medida de la información, se pasara la idea intuitiva que se tiene de ésta. En primer lugar se presentara que la cantidad de información proporcionada por cierto dato es menor cuanto más se espera ese dato, si una persona comenta el tiempo que hace en Bogotá, al decir "lluvioso" la información obtenida de este comentario es casi nula, ya es lo que se estaba esperando con mayor probabilidad, si por el contrario dice que es "soleado", se recibe más información, ya que la probabilidad de que esto ocurra es menor (Ver Tabla 1).

Es posible considerar las informaciones acerca del tiempo en Bogotá como datos que se reciben de cierta variable aleatoria, que cada vez que se interroga, arroja como respuesta el tiempo que hace en Bogotá. Esta variable aleatoria se convierte en una fuente de información.

La idea de la probabilidad de ocurrencia de cierto evento que arroja información queda ahora modelada por una variable aleatoria y su conjunto de mensajes y

probabilidades asociadas. Para el caso anterior, ver Tabla 1.

La ganancia de información que se experimenta, al recibir un mensaje, se puede entender como la reducción de incertidumbre sobre el estado de la fuente que se experimenta tras recibir el mensaje.

Tabla 1: Estado del tiempo y sus probabilidades asociadas

Tiempo	Probabilidad
Lluvioso	0.70
Soleado	0.30

1.2 FUENTES DE INFORMACIÓN

En este capítulo hablaremos acerca de las fuentes de información, estas pueden ser modeladas como una variable aleatoria; al recibir un mensaje de esa fuente, se obtiene una cantidad de información, que depende sólo de la probabilidad de emisión de ese mensaje; además, la cantidad de información es una función creciente con la inversa de la probabilidad (cuanta menor probabilidad, mayor cantidad de información se recibe, como se observó con en el ejemplo anterior “El clima en Bogota”).

Las fuentes de información se clasifican basándose en el tipo de señal que entregan y las variables independientes que determinan su estado, en este caso el tiempo es la variable independiente. De este modo se tienen los siguientes tipos de fuentes:

- Fuentes de tiempo continuo: la función está definida para cualquier valor de la variable independiente.
- Fuentes de tiempo discreto: la función sólo está definida para un conjunto contable de instantes de tiempo.

Pero se pueden clasificar también según el rango de valores que cubren las señales. En este caso los tipos de fuentes de información son:

- Fuentes continuas o de amplitud continua: el valor de la función toma un rango

continuo de valores.

- Fuentes discretas o de amplitud discreta: el valor de la función sólo toma un conjunto finito de valores. A cada uno de estos valores lo llamamos símbolo. El conjunto de todos los símbolos se suele llamar alfabeto. La elección del alfabeto es, en cierto modo arbitraria, ya que es posible agrupar varios símbolos para crear otros.

Estas dos clasificaciones son ortogonales. Es decir, existen fuentes continuas de tiempo continuo, fuentes continuas de tiempo discreto, fuentes discretas de tiempo continuo y fuentes discretas de tiempo discreto. Aunque en la práctica sólo se encuentran dos tipos: las llamadas Fuentes Analógicas, que son fuentes continuas de tiempo continuo; y las llamadas Fuentes Digitales, que son fuentes discretas de tiempo discreto. En esta monografía se centrara en el estudio de las fuentes digitales y en los últimos capítulos se abordaran las fuentes analógicas más adelante cuando se hable de imágenes, sonido, vídeo y texto.

Las fuentes digitales se clasifican según la relación que tenga un símbolo con los símbolos que han precedido su aparición. Así, las fuentes de información también se clasifican en:

- Fuentes sin memoria: los símbolos son estadísticamente independientes entre sí. Los símbolos que hayan aparecido hasta el momento no van a condicionar al símbolo presente ni a posteriores.
- Fuentes con memoria: la aparición de los símbolos no es estadísticamente independiente. Es decir, si han aparecido M-1 símbolos, el símbolo M-ésimo está condicionado por los anteriores.

1.2.1 ENTROPIA DE UNA FUENTE

En este capítulo se resaltara la importancia de la entropía de una fuente de información donde es significante resaltar la forma como se mide la misma, de acuerdo a la teoría de la información dicho nivel se puede medir de acuerdo a la entropía de la misma. Los estudios que se presentan con respecto a la entropía son de mucha importancia.

Dado una fuente “T” que transmite diferentes mensajes, donde se le está haciendo un control constante de las emisión de mensajes, se puede ver como resultado que los mensajes no resultan equiprobables sino que tienen un gran porcentaje de ocurrencia dependiendo de los mensajes emitidos; para codificar estos mensajes se procura utilizar menor cantidad de bits para que se logre una probabilidad y mayor cantidad de bits para los mensajes que obtienen menos probabilidad, todo esto para que el promedio de bits utilizados para codificar los mensajes sea menor a la cantidad de bits promedio de los mensajes originales. Esta es la base de la compresión de datos.

A este tipo de fuente se le puede denominar fuente de Orden-0 ya que la ocurrencia de los mensajes no depende de los mensajes enviados anteriormente y a las fuentes de orden superior se les representa mediante una fuente Orden-0 todo estos con las técnicas de modelación apropiadas.

Se define la probabilidad de una ocurrencia en un mensaje en la fuente como la cantidad de apariciones de dicho mensaje, donde se divide el total de mensajes; ahora supongamos que Pi (P sub i) se define como la probabilidad de la ocurrencia del mensaje-i de una fuente, y supongamos que Li es la longitud del código utilizado para representar a dicho mensaje, la longitud promedio de todos los mensajes codificados de la fuente se puede obtener como:

$$H = \sum_{i=0}^{n} Pi * Li$$

El promedio moderado del total de las longitudes en los códigos es de acuerdo a las probabilidades de ocurrencia, donde “H” se le puede denominar “Entropía de la fuente”. Esta entropía es de gran importancia ya que nos determina el nivel de comprensión de un conjunto de datos, ahora si consideramos un archivo como una fuente, de esta podemos obtener las probabilidades de ocurrencia de cada carácter del archivo, ya después de obtener la ocurrencia de cada carácter podemos calcular la longitud promedio del archivo comprimido, se demuestra que no es posible comprimir estadísticamente un mensaje/archivo más allá de su entropía.

Lo cual nos dice que considerando únicamente la frecuencia de aparición de cada uno de los caracteres, la entropía de la fuente no puede dar el límite teórico de

compresión, todo esto gracias a otras técnicas no estadísticas y puede este tal vez superar este límite.

El objetivo de toda compresión de datos es poder encontrar los Li que pueden minimizar a los ”H”, además estos deben poder determinar en función de los Pi, pues toda longitud de los códigos depende de la probabilidad de ocurrencia de estos, donde los más ocurrentes queremos codificarlos en menos bits.

Se plantea pues:

$$H = \sum_{i=0}^{n} Pi * f(Pi)$$

A partir de este punto y tras diferentes procesos matemáticos que los demostró Shannon, se llega a una conclusión de que H es mínimo cuando f(Pi) = log2 (1/Pi). Entonces:

$$H = \sum_{i=0}^{n} Pi * (-\log_2 Pi)$$

La longitud mínima con la cual puede codificarse un mensaje puede calcularse como Li=log2 (1/Pi) = -log2 (Pi). Todo esto nos podría dar una idea general de que la longitud a emplear en los códigos para los caracteres puede usar un archivo en función de su probabilidad de ocurrencia.

Reemplazando Li podemos escribir a H como:

$$H = \sum_{i=0}^{n} Pi * (-\log_2 Pi)$$

Para concluir se puede decir que la entropía de una fuente depende únicamente de la probabilidad de la ocurrencia de cada uno de los mensajes de la misma, por esto es importante los compresores estadísticos que son los encargados de la probabilidad de ocurrencia en cada carácter. Estos llevo a la conclusión a la oportuna de que no es posible comprimir una fuente de información más allá estadísticamente de nivel indicado por su entropía.

1.3 MODELO MATEMÁTICO DE LA TEORÍA DE LA INFORMACIÓN

La teoría de la información surgió finalizando la segunda guerra mundial, alrededor

de los años 40, nace con los trabajados de Claude Shannon 1948 y de Norbert Wiener en 1949, con estos trabajos se establecieron los fundamentos de la moderna teoría estadística de la comunicación. Ambos hombres investigaban la manera de extraer información a partir de un fondo ruidoso y ambos aplicaron conceptos estadísticos al problema.

Lo que se buscaba con este trabajo en su época era utilizar de manera más eficiente los canales de comunicación, donde fuera más óptima a la hora de enviar información, Wiener trato el caso en el cual, las señales portadoras de información están, en todo o en parte, fuera del control del diseñador y todo el procedimiento debe realizarse en el terminal receptor. El planteo el problema así: "dado un conjunto de posibles señales, no escogidas por nosotros, más el inevitable ruido, ¿cómo es posible hacer el mejor estimativo de los valores presentes y futuros de la señal recibida?". Las soluciones óptimas a este problema dieron origen a la denominada teoría de la detección.

El modelo propuesto consiste en un sistema general donde la comunicación como fuente de información, que atravesó de un transmisor, emite una señal, por el cual viaja por un canal, que puede verse afectado por ruido, posteriormente este llega a un receptor que decodifica el mensaje para entregarlo a el destinatario. Shannon planteo el problema de esta manera: “dado un conjunto de posibles mensajes que puede generar una fuente, no escogidos por nosotros, cual es la mejor forma de representar tales mensajes de tal manera que pueda transportar óptimamente la información sobre un sistema sujeto a las limitaciones físicas inherentes a los sistemas reales?”

Para tratar este problema de forma adecuada, es necesario concentrarse más en la información en sí que en las señales que la transporta. Para esto Shannon trato de hallar la repuesta a las siguientes preguntas:

- ¿En qué forma precisa es restringida la transmisión por las limitaciones físicas del medio? (Atenuación, Ancho de banda y Ruido).
- ¿Existe un sistema ideal de comunicación y, si es así, cuáles son sus

características?

- ¿Cómo pueden compararse los sistemas de comunicación existentes con el sistema ideal y como puede ser mejorado su comportamiento?

La teoría de la información es pues un cuerpo matemático que trata de responder a las preguntas anteriores a través de la definición precisa de tres conceptos básicos: la medida de la información, la capacidad de un canal de comunicación para transferir información, y la codificación como un medio para utilizar los canales al límite de su capacidad. Estos tres conceptos están enlazados en el que se ha dado en llamar el teorema fundamental de la teoría de la información, expresado así por C. Shannon y W. Wearver en su libro “The mathematical theory of comunication”:

"Dada la fuente de información y un canal de comunicación, existe una técnica de codificación tal que la información puede ser transmitida sobre el canal a cualquier velocidad inferior a la capacidad del canal y con una frecuencia arbitrariamente pequeña de errores a pesar de la presencia de ruido."

El aspecto más sorprendente y asombroso de este teorema lo constituye la posibilidad de transmisión libre de errores sobre un canal ruidoso por medio del uso de la codificación, pero advierte que la velocidad de transmisión se ve afectada por el ruido, lo cual es lógico pues en presencia de ruido se hace necesaria una codificación redundante de los datos.

En esencia la codificación se usa para acoplar la fuente y el canal con miras a lograr la máxima eficiencia en la transmisión de información, en forma análoga al acople de impedancias que busca la máxima transferencia de potencia.

A continuación se expondrán, uno de los conceptos de la teoría de la información:

Shannon y Weaver relacionan la cantidad de información de un mensaje con el grado de incertidumbre que este representaba para el receptor. Esto a su vez, se relaciona con la probabilidad de recibir un mensaje dado entre un conjunto de mensajes posibles, a más improbable el mensaje, mayor información con lleva para el usuario. Alternativamente en el lado transmisor, la medida de la información es

una indicación de la libertad de escogencia ejercida por la fuente al seleccionar el mensaje. Shannon postulo entonces que la medida de la información de un mensaje era función de la probabilidad de su aparición.

1.4 CANALES DE COMUNICACIÓN

Los canales de comunicación son el soporte que transmite información desde el emisor, el cual es el encargado de enviar el mensaje deseado hacia un receptor, el cual es el que lo recibe, todo esto se da gracias a un canal por el cual puede viajar una señal hacia su destino, está sujeta a tres fenómenos físicos: La atenuación, la distorsión y el ruido.

La señal por donde se transporta la información puede verte afectada o atenuada ya que por esta transita energía la cual se convierte en calor durante el proceso de trasmisión de información; también puede verse distorsionada debido a que el medio de transmisor tiene elemento lo cuales puede que un cambio en la energía almacenada requiere una cantidad definida de tiempo.

La velocidad a la que el medio puede cambiar su energía almacenada se determina a través de su respuesta de frecuencia la cual puede expresarse en términos del ancho de banda del sistema.

Como lo demostró Nyquist en su publicación de 1928, un canal de comunicaciones de ancho de banda W puede transmitir hasta 2W muestras independientes de la señal por segundo. Suponiendo que cada muestra corresponde a uno de m niveles posibles de señal. De aquí resulta que el número de bits por muestra es Log (m), por lo que el canal puede transmitir 2W Log (m) bps.

La velocidad de transmisión de símbolos del canal k, medida en sb/s, está directamente relacionada con la capacidad del canal y la codificación empleada para representar los mensajes emitidos por la fuente. Es posible concluir que la velocidad promedio a la que se transmite la información de una fuente por un canal, medida en bits/s. Depende de:

- La entropía de la fuente H(S). Es decir, de la información transmitida en

promedio con cada mensaje, medida en bits/msj.

- La longitud media n del código empleado para transmitir los mensajes, medida en sb/msj.

1.4.1 TIPOS DE CANALES DE COMUNICACION

Shannon a través de diferentes resultados matemáticos pudo establecer los recursos necesarios para la codificación óptima y para la comunicación libre de errores. Esto fueron aportes muy importantes para su época y aún más para la actual, donde tuvo aplicaciones en diferentes campos como lo fueron la radio, la televisión y la telefonía. De acuerdo a Shannon en 1948 nos dice que un sistema de comunicación general consta de varias parte, como primera instancia tenemos la fuente, que es la encargada de generar los mensajes para ser recibimos en un destinatario, por otra parte tenemos el transmisor, que se encarga de transformar el mensaje generado en una fuente de señal transmitida, en algunos casos donde esta información debe ser codificada este también es el encargado de codificarla. Un canal es el medio por el cual la señal se transmite desde un transmisor al receptor, algunos ejemplos de estos son los cables, fibra óptica o señal inalámbrica. Un receptor es el que se encarga de recibir la señal para posteriormente decodificarla e entregarla al destinatario.

Ilustración 1. Canal de comunicación.

Viendo el esquema, ahora veremos algunos ejemplos de tipos de canales de comunicación:

- *Par Trenzado*: Consiste en hilos de alambre de cobre torcidos en pares. La mayoría de los sistemas telefónicos en un edificio se apoyan en par trenzado, instalado para comunicación analógica.
- *El cable coaxial:* Consiste en un alambre de cobre con un gran espesor de aislamiento, que puede transmitir un mayor volumen de datos que el alambre torcido.
- *El Cable de figura óptica*: Consiste en haces de fibra de vidrio transparente, delgados, como un cabello humano, que se unen en cables.
- *Transmisión inalámbrica*: La transmisión inalámbrica envía señales a través del

aire o del espacio sin ninguna conexión física, y puede acompañarse de microondas terrestres, satélites, telefonía celular o rayos de luz infrarroja.

- *Microondas terrestres:* transmite señales de radio de alta frecuencia a través de la atmosfera y son ampliamente usados para las comunicaciones de alto volumen de largas distancias de punto a punto.
- *Los satélites:* se usan en general para las comunicaciones en instituciones muy grandes y geográficamente dispersas, que serían difícil enlazarlas mediante algún medio de cableado o microondas terrestres. Los satélites se mueven en orbitas estacionarias aproximadamente a 35mil kilómetros sobre la superficie de la tierra.

1.5 CODIFICACIÓN

1.5.1 Introducción

Un código permite acoplar la fuente y el canal para lograr una transmisión de información óptima. Del teorema fundamental de la teoría de la información se concluye que la salida de una fuente de mensajes puede ser codificada de tal forma que el número medio de bits por palabra de código sea igual a H, la entropía de la fuente. Se define entonces la eficiencia de un código como la relación (expresada en porcentaje) de la entropía H de la fuente y el número medio de bits por palabra de código.

La codificación de una fuente se realiza mediante la representación cada mensaje de la fuente por medio de una secuencia de símbolos de un alfabeto. Un alfabeto es un conjunto finito de símbolos, $A= \{a_1, a_2,..., a_n\}$. El alfabeto se elige de forma que sus símbolos se puedan transmitir eficiente y correctamente a través del canal.

En sistemas de comunicación digital usamos un alfabeto binario. Es decir:

$A = \{0, 1\}$

El conjunto de secuencias binarias asignadas a los mensajes de la fuente se denomina código, y cada elemento es una palabra código.

Un ejemplo es el código ASCII La fuente es el conjunto de caracteres ASCII {26 letras minúsculas, 26 letras mayúsculas, 10 dígitos numéricos y diversos caracteres especiales y signos de puntuación}. Las palabras código son secuencias de 7 símbolos binarios (Ejemplo: carácter ('a')=1000011).

Se presentara otro ejemplo. Dada la fuente S = {x_1, x_2, x_3, x_4} y el alfabeto A = {0, 1}, un posible código es el que se presentara en la Tabla 2.

Tabla 2: código ejemplo

x_i	x_1	x_2	x_3	x_4
$C(x_i)$	0	11	00	01

Un código se puede considerar como una aplicación inyectiva que asocia a cada mensaje una tupla de símbolos del alfabeto, ha de ser inyectiva para que a cada mensaje le corresponda una palabra distinta.

Según el número de símbolos del alfabeto, los códigos se clasifican en binarios, ternarios o en general, n-arios. Por ejemplo, el código ASCII es binario. El código Morse es ternario, pues su alfabeto consiste en A= {·,-, pausa}.

La longitud de una palabra código C_i es el número de símbolos que contiene (orden de la tupla), la denotamos como $L\ (C_i)$.

1.5.2 Clasificación de los Códigos

Los códigos se pueden clasificar de las varias formas dependiendo de ciertas características presentes en su formación.

- Códigos de longitud fija.

En la codificación de longitud fija, se asignan palabras de código de longitud iguales a cada símbolo en un alfabeto *A* sin tener en cuenta sus probabilidades. Si el alfabeto tiene *M* símbolos diferentes (o bloques de símbolos), entonces la longitud de las palabras de código en bits es el entero más pequeño mayor o igual que log_2

M.

Dos esquemas de codificación de longitud fija comúnmente usados son los códigos naturales y los códigos Gray. Puede demostrarse que la codificación de longitud fija sólo es óptima cuando:

- El número de símbolos es igual a una potencia de dos.
- Todos los símbolos son equiprobables.

Sólo entonces podría la entropía de la fuente ser igual a la longitud promedio de las palabras código que es igual a la longitud de cada palabra código en el caso de la codificación de longitud fija. A menudo, algunos símbolos son más probables que otros en este caso sería más ventajoso usar una codificación diferente a la longitud fija para aprovechar esta característica.

Es posible concluir que un código de longitud fija cumple con la siguiente expresión:

$$L(C_i) = L(C_j) \text{ para todo } i,j <= n$$

Por ejemplo, el código ASCII es un código uniforme de longitud de palabra 7. Para un código uniforme, L_m es la longitud constante de palabra.

- Códigos de traducción única.

Al recibir la secuencia de símbolos correspondiente a una secuencia de mensajes, ésta puede decodificarse sin ambigüedad, esta clase de códigos también son llamados unívocamente decodifícable.

Tabla 3. Ejemplo de código no unívocamente decodifícable

x_i	x_1	x_2	x_3	x_4
$C(x_i)$	1	10	11	01

Porque existe al menos una secuencia de símbolos que admite más de una

decodificación: 1011 puede decodificarse como x_2x_3, $x_1x_4x_1$ o $x_2x_1x_1$.

Tabla 4. Ejemplo de código unívocamente decodifícable

x_i	x_1	x_2
(x_i)	0	01

El símbolo 1 hace de delimitador. Por ejemplo, la secuencia 0010001 sólo admite la decodificación $x_1x_2x_1x_1x_2$.

- Código instantáneo

Es posible decodificar al terminar de recibir cada palabra, sin necesidad de esperar a las palabras que le suceden Esto ocurre si y sólo si ninguna palabra es prefijo de otra (código prefijo).

Tabla 5. Ejemplo de código no instantáneo

x_i	x_1	x_2	x_3	x_4
$C(x_i)$	0	01	110	1110

La secuencia 0110 sólo admite una decodificación: x_1x_3, pero hay que esperar al final de la segunda palabra para poder decodificar la primera.

Tabla 6. Ejemplo de código instantáneo

x_i	x_1	x_2	x_3	x_4
$C(x_i)$	0	10	110	1110

Ejemplo, suponiendo que se emite la secuencia de mensajes $x_1x_3x_4$, codificada como 01101110. Es Posible decodificar la secuencia 01101110 al final de cada palabra.

Teorema 1.- Todo código C uniforme es instantáneo. El reciproco no es cierto.

Tabla 7. Ejemplo de código instantáneo y no uniforme

x_i	x_1	x_2	x_3	x_4

$C(x_i)$	0	10	110	1110

Teorema 2.- Todo código C instantáneo es de traducción única. El reciproco no es cierto.

Tabla 8. Ejemplo de código de traducción única y no instantáneo

x_i	x_1	x_2
$C(x_i)$	0	01

- Código óptimo

Dada una fuente S y un alfabeto A, decimos que un código C es óptimo si y sólo si C es instantáneo y no existe otro código instantáneo con menor longitud media. Los códigos óptimos no son únicos.

Antes de estudiar cómo construir códigos óptimos, se estudiara cómo construir códigos instantáneos y una condición necesaria y suficiente para la existencia de estos.

Teorema 3.- (Desigualdad de Kraft-McMillan) Sea un código binario C con m palabras de longitudes $n_1, n_2, \ldots n_m$. Si C es instantáneo, entonces verifica necesariamente:

$$K(C) = \sum_{i=1}^{m} 2^{-n_i} \leq 1$$

Tabla 9. Ejemplo de un código instantáneo *C*

x_i	x_1	x_2	x_1	x_2
$C(x_i)$	0	01	110	1111
n_i	1	2	3	4

Por ser instantáneo, debe verificar la desigualdad de Kraft-McMillan. En efecto:

$K(C)= 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} = 15/16 <= 1$

Teorema 4.- Dado un conjunto de enteros $n_1, n_2, \ldots n_m$, que satisfacen la desigualdad de Kraft-McMillan, siempre es posible encontrar un código instantáneo C cuyas longitudes de palabra son $n_1, n_2, \ldots n_m$. El teorema establece una condición suficiente para la existencia de un código instantáneo con unas longitudes de palabra dadas. Tanto este teorema, como el anterior, son también válidos para códigos n-arios en general.

1.5.4. Tipos de codificación

Existen diferentes tipos de codificación y cada una tiene aplicaciones específicas. A continuación se describirán cada uno de los diferentes tipos de codificación.

- Codificación en la fuente

El objetivo de la codificación es obtener una representación eficiente de los símbolos del alfabeto fuente. Para que la codificación sea eficiente es necesario tener un conocimiento de las probabilidades de cada uno de los símbolos del alfabeto fuente.

El dispositivo que realiza esta tarea es el codificador de la fuente. Este codificador debe cumplir el requisito de que cada palabra de código debe decodificarse de forma única, de forma que la secuencia original sea reconstruida perfectamente a partir de la secuencia codificada.

- Codificación del canal

En ocasiones se producen diferencias entre las secuencias de datos enviadas a través de un canal y las secuencias de datos recibidas debidas a la existencia de ruido en el canal. A estas diferencias se les denomina errores. Por ello es necesario realizar una codificación a la entrada del canal, cuyo objetivo es que el receptor sea capaz de detectar y corregir los errores producidos en los datos durante su transmisión por el canal.

La codificación del canal consiste en introducir redundancia, de forma que sea

posible reconstruir la secuencia de datos original de la forma más fiable posible. Aun aplicando una codificación del canal redundante, los errores pueden aparecer es allí donde se hace necesario aplicar las llamadas técnicas de detección y corrección de errores. Se presentara en qué consisten cada una de estas técnicas:

- Detección de Errores o Corrección Hacia Atrás (ARQ - Automatic Repeat Request): cuando el receptor detecta un error solicita al emisor la repetición del bloque de datos transmitido. El emisor retransmitirá los datos tantas veces como sea necesario hasta que los datos se reciban sin errores.
- Corrección de Errores o Corrección Hacia Delante (FEC - Forward Error Correction): se basa en el uso de códigos autocorrectores que permiten la corrección de errores en el receptor.

2 ALMACENAMIENTO DE DATOS EN SISTEMAS DIGITALES

2.1 INTRODUCCIÓN

Los datos digitales mueven la sociedad actual, la enorme cantidad de información que se incorpora cada día a la Internet es sólo una cara de la cuestión: también están las tarjetas de crédito, los informes y registros bancarios, el campo de las comunicaciones telefónicas y de la administración fiscal, los datos generados en el mundo laboral, las imágenes, la música y muchos más. En todos estos campos se generan montañas de información que hay que conservar en algún sitio.

El almacenamiento de información ha dejado de ser algo simplemente implícito en los sistemas actuales y paso a reclamar un papel central y protagonista, en los planes y estrategias informáticas en el ámbito doméstico y empresarial.

En el mercado de centros de datos, los proveedores operan en un sector en constante evolución, los cambios en este sector son de tal magnitud que todavía no se han encontrado modelos empresariales estables para el mismo.

2.1.1 Evolución de tecnología del almacenamiento masivo

Antes de la aparición de las primeras computadoras comerciales electrónicas en 1951, almacenamiento en “masa” aunque pequeño comparado con los estándares actuales era una necesidad. A mediados de 1800, se usaron *punch cards* para proporcionar la entrada a máquinas calculadoras, en la década de los 1940 introdujo el uso de los tubos de vacío para el almacenamiento, hasta que finalmente una cinta de papel comenzó reemplazar a las *punch cards* alrededor 1950. Sólo un par de años más tarde, los medios magnéticos aparecieron en la escena, y en 1957 se introdujo como un componente de IBM RAMAC la primera unidad de discos duros 350.Esta unidad Requirió 50 discos de 24 pulgadas para guardar cinco megabytes (millón bytes, se abrevió MB) de datos y costó bruscamente US$33.000 por año o arrendarlo a US$7.000 por megabyte anual.

Los altos costos de los discos duros, fueron la principal causa para que los primeros PCs no incluyeran en su interior una unidad rígida de almacenamiento, estos solo disponían de una o dos unidades de disco flexible, allí se almacenaba el sistema operativo, y los programas que serían usados junto con los archivos de trabajo generados por el usuario, de esta forma los primeros usuarios de las PCs se las tenían que arreglar para confinar su información de trabajo en un medio reducido en capacidad y prestaciones, sin dejar de mencionar su baja confiabilidad.

Con la revolución de la computadora personal a comienzos de 1980 cambió todo, es ahora cuando se introducen los primeros discos duros pequeños. Eran discos de 3.25 pulgadas los que manejaban de 5 a 10 MB de almacenamiento el equivalente de 2.500 a 3.000 páginas de doble espacio de tecleo de información en un aparato del tamaño de la caja de un zapato pequeño. No es de sorprender que en este tiempo se considerara que una capacidad de almacenamiento de 10 MB era demasiado grande para una llamada computadora "personal".

Los primeros PCs usaron discos flexible portables como aparatos de almacenamiento casi exclusivamente. El término "disco blando" con precisión se refiere a los primeros discos para PC de 8 y 3.25 pulgadas que tuvieron éxito. Los discos internos de hoy, más pequeños, se construyen 3.5 pulgadas de forma similar a los anteriores, pero se albergan en un casco de plástico rígido, que es más

durable que el techado flexible de los discos más grandes.

Con la introducción del IBM PC/XT en 1983, el disco duro también volvió a ser un componente normal de computadoras personales. La descripción "duro" se usa porque los discos internos que contienen los datos se sostienen en una unidad de aluminio rígido que los liga. Estos discos, se cubren con un material magnético de mayor duración y calidad que el plástico utilizado en los discos blandos. La vida útil de una unidad de discos duros, está en función de la unidad de disco que lee/escribe (cabeza): en un disco duro, las cabezas no tienen un contacto directo con la unidad de almacenamiento, por el contrario en un disco blando la cabeza que lee/escribe está en directo contacto, con lo que causa un deterioro con el uso.

2.2 TECNOLOGÍAS DE ALMACENAMIENTO DIGITAL DE DATOS

Dado el incremento de las necesidades de los usuarios, los fabricantes se han visto en la necesidad de realizar inversiones en tecnología para diseñar dispositivos de almacenamiento con mayores capacidades y más sofisticados.

En este momento el mercado ofrece gran diversidad de equipos de almacenamiento entre los que se pueden mencionar:

- Discos Flexibles
- Discos Duros
- CD-ROM, CD-RW
- DVD
- Memorias Flash

A continuación se presentan los diferentes dispositivos que actualmente se utilizan, se mencionaran sus ventajas, desventajas y durabilidad para el almacenamiento.

2.2.1 Medios de almacenamiento magnético

Dejando de lado la distinción entre cintas o discos, el almacenamiento magnético de datos binarios consiste, en definitiva, en imprimir un patrón de magnetización

sobre un medio. Ello corre a cargo de un cabezal de grabación inductiva, que consta de un filamento conductor bobinado varias veces alrededor de un núcleo anular que puede magnetizarse fácilmente. La corriente eléctrica del conductor induce un fluido magnético en el núcleo; invertido el sentido de la corriente se invierte la dirección del flujo magnético. Puesto que las líneas de flujo magnético se propagan a medida que van salvando intervalos del núcleo anular (también llamados "entrehierros" o "*graps*") pueden servir para magnetizar un medio situado en la vecindad del el entrehierro. La dirección de la magnetización depende de la dirección de las líneas de flujo magnético que "puentean" el entrehierro.

Así pues, un patrón de magnetización se ira imprimiendo en el medio, conforme este se mueve bajo el entrehierro, con la mera inversión repetitiva del flujo de corriente del cabezal de grabación. Ahora bien. Los datos generados por computadora se almacenan en cadenas de dígitos binarios; en consecuencia, el patrón de inversiones de corriente (y por ende el patrón de inversiones de magnetización) corresponderá a un sucesión de ceros y unos que constituyen los bits de datos.

Si se trata de un disco, los bits de datos se graban normalmente en pistas circulares concéntricas. Hasta cierto punto, el número de bits que cabe en una pista (la densidad lineal de bits) puede aumentarse menguando las regiones de transición, y por consiguiente, facilitando la introducción de más regiones magnetizadas en las pistas. A pesar de ello, la dimensión mínima de una región de transición tiene un límite.

- Discos flexibles.

Es un disco impregnado de material ferromagnético, están protegidos con una carcasa de plástico duro.

Para realizar el proceso de lectura / escritura el disco se tiene que enganchar a un motor que gira a 300 - 360 revoluciones por minuto (r.p.m.), un segundo motor lineal que controla un brazo con la cabeza de lectura / escritura que describe un

movimiento radial respecto al disco. Toda la información del disco se puede acceder con movimientos en 2D. La grabación en este medio se realiza por inducción electromagnética.

El cálculo del tiempo de acceso en el peor caso considera el tiempo de giro completo (170 - 200 ms) y el tiempo de desplazamiento radial completo del brazo (200 ms); de esta forma el tiempo de acceso en el caso peor es de 400 ms. Como todos los medios de almacenamiento los discos flexibles tienen ventajas y desventajas, entre las ventajas encontramos la portabilidad y el costo, sus desventajas son la seguridad y protección de los datos.

A pesar de sus desventajas el disco flexible sigue siendo hoy un medio de almacenamiento y transporte de datos, además sus unidades de lectura / escritura siguen siendo un compañero inseparable de las computadoras actuales.

- Discos rígidos.

Estos dispositivos son más potentes en capacidad de almacenamiento (100 Sec/Track) y velocidad de transmisión (7200 r.p.m.) que los discos flexibles. Compuestos por 2 a 8 discos individuales de aluminio más ligeros que de hierro lo disminuye su inercia y facilita la aceleración, giran como un cuerpo sólido y consta de 4 a 16 cabezas de lectura / escritura.

El formato más popular es de 3.5 pulgadas; la capacidad física depende de los métodos de grabación y de codificación. La región más pequeña magnetizable depende tanto del tamaño de la cabeza como de su distancia al disco, las cabezas flotan sobre el disco a 1 o 0.5 mm. Un brazo rígido con motor lineal facilita que todas las cabezas se muevan juntas y desplaza las cabezas a través del radio del disco.

La capacidad generalmente se expresa en GB (Gigabyte) y difiere de un disco con o sin formato en un 20 %aproximadamente. El tiempo de acceso promedio es igual a latencia más tiempo promedio de posicionamiento. En los discos

actuales la latencia es aproximadamente 4 ms y el tiempo promedio de posicionamiento es aproximadamente 65 ms, el tiempo Track-Track, desplazamiento y asentamiento es aproximadamente 2.5 ms.

La Tasa de Transferencia de Datos (TTD) se calcula así:

TTD=(Sectores/Track * bytes/sector * r.p.m.)/Interleave.

El tiempo que se tarda en transferir un bloque de datos desde el disco al controlador de disco se divide en:

T Dato-controlador = T transferencia Datos + T acceso

Por último es posible decir que este medio tiene un costo bajo por GB y entre sus características más importantes se destaca la fiabilidad (Tiempo Medio entre Fallos MTBF ~ 17 años).

2.2.2 Medios ópticos

Los discos ópticos son dispositivos de almacenamiento para grandes sistemas electrónicos de archivos. Tiene como diferencia contra otros métodos de almacenamiento que este es no magnético. Estos dispositivos son utilizados para trabajar con grandes bases de datos, con aplicaciones que requieren almacenamiento de archivos de voz y vídeo, mediante el cual se pueda almacenar, localizar, transmitir, procesar y gerenciar documentos. Existen diferentes tipos de unidades de discos ópticos: Unidades de lectura, unidades de lectura y una sola escritura y las unidades de lectura y escritura. Las unidades de solo lectura, son unidades que como su nombre lo indica sirven solamente para la lectura de discos pregrabados o del fabricante, estos discos tienen una capacidad de almacenamiento entre 650MB y 700MB.

- Unidad de CD-ROM.

En este dispositivo, a diferencia de los discos magnéticos, el movimiento del disco es variable. Con esta característica se pretende conseguir que la velocidad lineal de

lectura sea constante en todos los puntos, sean interiores o exteriores. Son dispositivos solo de lectura, pero a partir de ellos se han desarrollado nuevos soportes compatibles, llamados CD-R (permiten ser escritos solo una vez) y CD-RW (Permiten ser escritos y luego borrados para usarlos de nuevo), ambos tanto CD-R como CD-RW comparten la misma capacidad de almacenamiento.

Este medio en lugar de disponer su superficie en pistas circulares concéntricas lo hace en una única pista que comienza en el centro del disco y crece en espiral hasta el borde de la superficie circular. Esta pista está distribuida en sectores de igual tamaño.

Las unidades de lectura / escritura para este medio constan un motor que rota el CD variando la velocidad rotacional para mantener la velocidad lineal constante, un dispositivo de lectura que consiste en un diodo láser, un prisma y un diodo sensible a la luz, además de lentes montadas sobre un dispositivo de enfoque.

En su funcionamiento, el láser pasa directamente a través del prisma y proyecta un haz enfocado sobre la superficie del CD, el láser penetra en la capa protectora e incide sobre la capa reflexiva de aluminio de la parte superior del CD, que consiste en una serie de “Lands” (*superficies planas*) y de“ Pits” *perforaciones*, estas superficies son el registro en código binario usado para almacenar la información; cuando la luz que alcanza un “Pit” es dispersada, la que incide en un “Land” es reflejada hacia el diodo fotosensible, cada pulso de luz que alcanza el fotodiodo genera una señal eléctrica que será entendida como código binario.

- DVD.

Este medio intenta servir de soporte universal para el almacenamiento de información informática y datos multimedia, reemplazando al CD.

El proceso de lectura es básicamente el mismo que el caso de los CD-ROM, tiene las mismas dimensiones y es también existe el DVD-R y DVD-RW; lo diferencia del CD una velocidad de transferencia muy superior, formatos de grabación distintos,

la utilización de las dos caras para almacenar la información y por último la diferencia más importante, su mayor capacidad de almacenamiento, la cual se debe a una reducción del tamaño de los pozos y el espaciado entre pistas, láser de longitud de onda menor (635/650 nm vs 780 nm del CD-ROM) y la posibilidad de que en una misma cara se sitúen dos capas de información separadas por un material semi-reflexivo.

2.2.3 Dispositivos de estado sólido

Son componentes electrónicos basados en semiconductores. La capacidad de estos dispositivos hoy día supera la capacidad de los pequeños discos rígidos con los cuales vienen equipados los dispositivos portátiles.

- Tarjetas de memoria Flash.

A medida que la popularidad de las cámaras digitales, reproductores MP3, PDA y otros dispositivos portátiles ha aumentado, también lo ha hecho la necesidad de dispositivos de memoria más pequeños y baratos. La memoria flash se destaca dentro de esta nueva generación de dispositivos de almacenamiento. Este dispositivo usa circuitos integrados de estado sólido para guardar sus archivos de imagen, sonido, vídeo y en general cualquier clase de datos.

La memoria flash es un chip semiconductor, que permite varios ciclos grabado/borrado aplicándole una carga eléctrica. A pesar de que las memorias flash son similares a los chips de memoria RAM usados dentro de la computadora, hay una diferencia importante, esto se debe a que una memoria Flash no requiere ninguna batería para mantener almacenada la información y no pierden la información almacenada cuando el suministro eléctrico del dispositivo ha finalizado. La información almacenada en este medio se mantiene indefinidamente sin necesidad de suministro eléctrico alguno a los componentes de memoria flash.

En su aspecto físico la memoria Flash no es más que un chip empaquetado dentro de una cubierta y equipado con conectores eléctricos, a esta unidad sellada en su conjunto se le llama tarjeta.

Las tarjetas de memoria flash consumen poca energía, ocupan poco espacio(portables), y son muy resistentes mecánicamente; debido a sus componentes de estado sólido y a la ausencia de partes móviles que salten o se rompan, estas características las hace altamente confiables.

Estas tarjetas vienen en una variedad de formatos que no son intercambiables entre sí. Una vez que usted tiene hecha una inversión apreciable en un tipo de tarjetas de memoria, usted se encontrará en la opción de usar sólo los dispositivos que soportan su formato.

Hasta recientemente, la mayoría las tarjetas flash han estado en el estándar de tarjeta PC Card (PCMCIA) formato que se usa ampliamente con computadoras portátiles. Sin embargo, con el crecimiento de la cámara digital, reproductores MP3 y otros mercados, se han introducido varios formatos más pequeños.

Como resultado de la competencia, los dispositivos portables soportan una confusa variedad de tarjetas de memoria flash incompatibles entre sí que incluyen, pero no limitadas a lo siguiente:

- Tarjetas de PC
- CompactFlash
- SmartMedia
- MemorySticks
- Memorias USB

Cada uno de estos formatos tiene capacidades de almacenamiento que oscilan entre 2MB y 3GB, con velocidades que están alrededor de 4,5 MB/s en la lectura y 3,2MB/s de escritura.

3 TEORÍA DE LA COMPRESIÓN DE DATOS

3.1 INTRODUCCIÓN

Comprimir significa reducir el tamaño de algo. El objetivo principal de cualquier esquema de compresión es describir la misma información con un conjunto de datos de menor magnitud. Pero, ¿por qué es importante reducir la magnitud de los datos necesarios para representar una información determinada?

La primera razón en obvia: para poder colocarlos en un espacio menor. Actualmente existen dispositivos con una gran capacidad de almacenamiento (Discos Duros, USB, etc.) pudiendo guardar en ellos muchos datos de gran tamaño. Sin embargo, la complejidad y volumen de los datos aumenta en forma casi paralela al aumento de las capacidades de estos medios de almacenamiento. Además si se desean transportar datos en otro medio (Memorias USB, CD-ROM, DVD), el espacio está más limitado, y puede no ser suficiente para contener los datos que se requieren transportar. Por ello hay que comprimirlos.

La segunda razón aparece fundamentalmente con la redes de comunicaciones, la Internet es un ejemplo de ello. Esta segunda razón es el tiempo (que también se traduce en dinero) que es necesario para acceder o descargar archivos de gran tamaño por la red. Por tanto, si esos datos están comprimidos se tardara menos (y se gastara menos dinero) en enviarlos o recibirlos (por correo electrónico, web, ftp o cualquier otro protocolo de transferencia de datos).

Lógicamente la compresión suele llevar asociada una descompresión, con la que se pueden recuperar el formato original de esos datos para poder utilizarlos. Dependiendo de la forma en que se lleve a cabo el proceso de compresión/descompresión, los compresores se pueden clasificar tomando como base los siguientes factores:

Según el tiempo de necesario para comprimir / descomprimir:

- Simétricos: en este tipo de compresores el proceso de descompresión consume una cantidad de tiempo igual o muy cercana a la se gastó en el

proceso de compresión. Este tipo de compresores es de uso general en los procesos que implican transmisión de datos en tiempo real. Por ejemplo el vídeo conferencias, voz sobre IP y aplicaciones de telemetría.

- Asimétricos: en este tipo de compresores el tiempo de compresión y descompresión difieren en un alto margen, generalmente el proceso de compresión tarda mucho más que el de descompresión. Estos compresores son utilizados generalmente para comprimir grandes volúmenes de datos, con la característica de que el proceso de compresión generalmente se realiza solo una vez, mientras que el de descompresión se lleva acabo muchas veces. Por ejemplo, la compresión de vídeo, el proceso de compresión es muy lento y si no se cuenta con una máquina de buen rendimiento el tiempo que tarda puede ser excesivo, pero con la ventaja de que solo se realiza una vez, después se almacena el vídeo comprimido en un medio y desde allí se puede reproducir tantas veces como sea necesario. Con la ventaja que implica un proceso de descompresión de relativa baja complejidad en comparación con el proceso de compresión.

Según el modelo utilizado los compresores se clasifican en:

- No Adaptativo: En este tipo de compresores el modelo se mantiene constante y ha sido predefinido previo al proceso de compresión. Estos compresores son rápidos, pues no es necesario un reconocimiento previo de los datos a comprimir. La implementación del algoritmo de compresión es muy sencillo pero la razón de compresión generalmente es muy baja.
- Semi-Adaptativo: los compresores semi-adaptativos definen el modelo con base a un examen previo de los datos. Es decir, se establece un modelo basado una mirada panorámica del total de datos a comprimir y después se aplica dicho modelo de forma invariable en el proceso de compresión. Para el funcionamiento de estos compresores se requiere que los datos a comprimir estén almacenados o que su transmisión se lleve acabo dos veces (en la primera transmisión se establece el modelo y en la siguiente se aplica dicho modelo para realizar la codificación). Esto es un inconveniente ya que los datos no pueden ser tratados como una corriente.

- Adaptativo: este tipo de compresores se caracteriza por la actualización dinámica del modelo. Es decir, conforme se realiza la compresión la modelo aplicada varía dependiendo del comportamiento de los datos de entrada.

Existen dos ventajas principales de estos compresores sobre los anteriores. En primer lugar, se realiza un único recorrido de los símbolos codificados. Aunque esto pueda parecer una simple mejora, en determinadas situaciones es esencial ya que la secuencia no tiene que ser almacenada para ser comprimida, y así, dicho proceso puede realizarse simultáneamente a la llegada de los símbolos al compresor (la secuencia de símbolos puede ser tratada como una corriente).

En segundo lugar presentan mejores razones de compresión, esto se debe, principalmente, a que la codificación de los símbolos varía durante el transcurso de la compresión. Así, la longitud de los códigos que son asignados los símbolos, aumenta o disminuye según las variaciones de los patrones del modelo, los cuales están directamente ligados a los datos de entrada y su comportamiento en el tiempo.

Los compresores adaptativos también tienen características no muy buenas, entre las cuales se destacan dos:

En primer lugar son más lentos que los anteriores. En segundo lugar su implementación es más compleja, pues deben resolver dos problemas:

a) La actualización del modelo
b) La inclusión de nuevos símbolos en el modelo.

La compresión de datos ahorra espacio de almacenamiento y reduce el ancho de banda necesario para la transmisión de datos; pero a cambio consume tiempo, recurso que generalmente no se está dispuesto a pagar, es por eso que todos los datos no se almacenan o transmiten comprimidos.

Para continuar hablando de compresión se debe establecer una diferencia entre

información y *datos*, ya que en muchas ocasiones se utilizan como sinónimos y no lo son. Los datos son una forma representar la información; así, una misma información puede ser representada por distintas cantidades de datos, por tanto, algunas representaciones de la misma información contienen datos redundantes. A continuación se presentan algunas definiciones de compresión de datos:

- La compresión consiste en tomar una trama de símbolos y transformarlos en códigos / claves. Si la compresión es eficiente, las claves resultantes ocuparán menor espacio que los símbolos originales.
- La compresión de datos se define como el proceso de reducir la cantidad de datos necesarios para representar eficazmente una información. Es decir, la eliminación de datos redundantes.

La redundancia es una característica de los datos, que hace que los símbolos de la fuente de datos no tengan un carácter aleatorio, pudiendo en determinadas circunstancias, determinarse una curva no uniforme de distribución de símbolos.

Tipos de redundancia:

- Repetición de caracteres
- Repetición de patrones
- Repetición de cadenas
- Distribución de caracteres

En el caso de las imágenes, existen tres tipos de redundancia:

- Código redundante
- Píxeles redundantes
- Redundancia visual

Proceso de compresión. En el proceso de compresión existen dos elementos fundamentales un modelo y un codificador. Un modelo es simplemente una colección de datos y reglas usados para procesar los datos de entrada, es aquí

donde se extrae información sobre redundancias existentes en los datos y se describen en forma de modelo.

Como no suelen existir modelos exactos, para mantener la correspondencia entre el modelo y los datos reales se debe generar una descripción del modelo y una descripción de las diferencias entre los datos reales y el modelo, a lo que llamaremos residuo.

La compresión se puede hacer de dos formas:

- Sin Perdida: manteniendo la integridad de la información. Es decir, que al descomprimir obtenga exactamente lo que tenía en un principio, sin perder nada de información en el proceso.
- Con Perdida: sin mantener esa integridad, pudiendo perder ciertos datos que sean menos relevantes y que no se podrán recuperar más tarde al descomprimir.

Para comprimir los datos se utilizan distintos algoritmos, estando estos especialmente diseñados para cada tipo de dato. El resultado de la compresión será un archivo en un formato específico, de menor o igual tamaño que el original, dependiendo de qué tan eficiente sea el algoritmo de compresión, se puede presentar el caso en el cual el archivo compreso es de mayor tamaño que el archivo original, esto generalmente se debe a que se aplicó un algoritmo de compresión no adecuado; Es decir el algoritmo aplicado no explota la redundancia asociada a los datos de entrada, cabe decir que existen varios tipos de redundancia y que los datos pueden poseer uno o varios tipos de redundancia asociados, dependiendo de su formato.

3.2 COMPRESIÓN SIN PERDIDA

3.2.1 Compresores estadísticos.

Los compresores estadísticos son los que utilizan la información de las probabilidades de los mensajes de la fuente para construir una codificación.

No parece extraño que habiendo usado una medida de información basada en las probabilidades se encontrara con compresores que utilicen las propiedades estadísticas de la fuente de información para mejorar la codificación (el conjunto de símbolos de salida asociados a cada mensaje emitido por la fuente) de los mensajes de la fuente. Estos son los compresores estadísticos.

En estos compresores, se pueden utilizar las siguientes variantes del modelo estadístico:

- No adaptativo: en este tipo de compresores, las probabilidades de los símbolos son constantes. Son rápidos, pues solo se realiza una sola pasada sobre los datos, pero la razón de compresión es muy baja. El algoritmo de compresión es muy sencillo.
- Semi-adaptativo: los compresores semi-adaptativos realizan dos pasadas sobre la secuencia de símbolos a codificar. En la primera pasada se calculan las probabilidades asociadas y en la segunda se realiza la codificación.
- Adaptativo: Las probabilidades instantáneas de los símbolos varían conforme se produce la compresión. Se realiza un único recorrido de los símbolos codificados. Así, la longitud de los códigos que son asignados los símbolos, aumenta o disminuye según la frecuencia de aparición de estos en el tiempo.

Entre los compresores que usan un modelo estadístico es posible encontrar varios tipos:

a) Compresores del tipo Huffman ó Shannon-Fano

b) Compresores Aritméticos

c) Compresores Predictivos

d) Compresores Huffman

Estos compresores utilizan un modelo estadístico y un codificador Huffman, de allí su nombre. La codificación de Huffman fue descrita por primera vez por David A. Huffman en 1952 en un artículo llamado “A Method for the Construction of Minimum

Redundancy Codes". Debido a su facilidad de cómputo, es ampliamente utilizada en programas de compresión, máquinas de fax y en esquemas de compresión de imágenes como JPEG. Los compresores Huffman presentan las siguientes características:

a) Usa códigos de longitud variable, asignando a los símbolos más frecuentes un código de compresión de menor longitud a costa de asignar códigos de mayor longitud a los menos frecuentes.

b) El número de bits por código de compresión es un número entero. Esto provoca que si la entropía de un símbolo es 2.5 bits, el código de Huffman asignará un código con una longitud de 2 o 3 bits, pero no 2.5 bits. Debido a esto, la codificación de Huffman no puede ser considerada como un método de codificación óptimo, pero si es la mejor aproximación para construir códigos de compresión con un número entero de bits. En consecuencia, sería imposible la codificación eficiente de fuentes binarias (que sólo generan dos símbolos) como es el ejemplo de una imagen binaria (que sólo tiene dos tonos). Por suerte, en estos casos la extensión de la fuente es la solución ya que, en general, el número de símbolos-bloque (formados por la concatenación de símbolos) aumentará respecto del número de símbolos, y si la fuente no es aleatoria, una representación de longitud variable siempre será conveniente.

c) Aunque los códigos asociados a símbolos diferentes tienen diferente longitud, éstos pueden ser decodificados de forma única e instantánea. Esto debido a que en la codificación Huffman se usan códigos instantáneos. Del capítulo 2 sabemos que dichos códigos se basan en que ningún código es prefijo (forma parte inicial) de otro.

Codificación Huffman. Para construir un código de Huffman normalmente se utiliza un árbol binario que se construye a partir de las probabilidades de los símbolos que deseamos codificar. El diseño de dicho árbol se realiza de la siguiente forma:

Paso 1. Crear tantos nodos (que serán las hojas del árbol) como símbolos vayamos

a codificar. En cada nodo se almacena el símbolo codificado y su probabilidad. Estos nodos forman la lista de nodos sin procesar.

Paso 2. Extraer dos nodos de la lista de nodos sin procesar que tengan una probabilidad mínima.

Paso 3. Insertar en la lista un nodo que sea padre de los dos nodos seleccionados. Este nodo no tiene asociado ningún símbolo y su probabilidad será la suma de las probabilidades de los nodos hijo.

Paso 4. Repetir los pasos 2 y 3 hasta que únicamente se tenga un nodo en la lista de nodos sin procesar, nodo que debe tener una probabilidad igual a 1 y que se llama nodo raíz del árbol de Huffman.

Puesto que en cada iteración se extraen dos nodos de la lista y se inserta otro, el número de iteraciones necesarias para construir un árbol con N hojas (el número de símbolos que se desean codificar) es N. Una vez construido el árbol, el código de compresión asociado a un símbolo se calcula recorriendo el árbol desde la raíz hasta el símbolo codificado (situado siempre en una hoja). Se presentara un ejemplo de codificación.

Ejemplo: Una cadena con 100 caracteres. Se sabe que aparecen 6 caracteres diferentes y la frecuencia de aparición de cada uno de ellos está dada en la tabla 10.

Tabla 10. Frecuencia de aparición de cada carácter

Carácter	a	b	c	d	e	f
Frecuencia	40	25	15	10	5	5

Para dar solución a este ejemplo se seguirán cada uno de los pasos mencionados anteriormente para la construcción del árbol.

Paso 1: Inicialmente se crea un nodo por cada símbolo que se va a codificar y se inserta

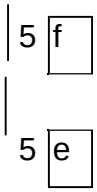
en la lista de nodos sin procesar, en este caso son 6 nodos.

Ilustración 2. Lista de nodos sin procesar

A	b	c	D	E	f
40	25	15	10	5	5

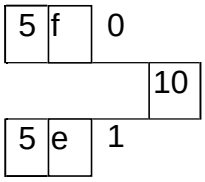
Paso 2: se extrae de la lista de nodos sin procesar los dos nodos con frecuencia mínima, es válido recordar que los nodos se extraen sin reemplazo. Es decir, una vez estos son seleccionados se eliminan de la lista de nodos sin procesar.

Ilustración 3. Paso 2 - nodos con frecuencia mínima



Paso 3: se crea un nuevo nodo en la lista de nodos sin procesar el cual es no tiene asociado ningún símbolo y su frecuencia es igual a la suma de las frecuencias de sus dos nodos hijos.

Ilustración 4. Paso 3. Inserción de un nuevo nodo



Paso 4: Aún quedan en la lista de nodos sin procesar los siguientes nodos:

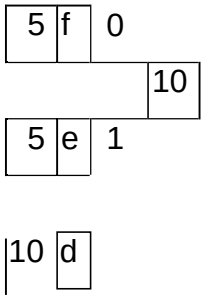
{[a,40],[b,25],[c,15],[d,10],[Padre(f,e),10]}

Por lo tanto se deben repetir los pasos 2 y 3 nuevamente.

Segunda iteración. Pasó 2: se extrae de la lista de nodos sin procesar los dos nodos con frecuencia mínima, en esta iteración los dos nodos con frecuencia mínima corresponden a:

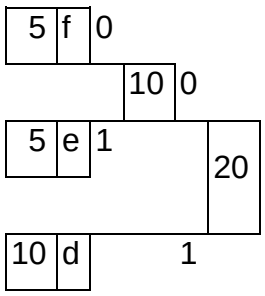
{[d,10],[Padre(f,e),10]}

Ilustración 5. Segunda iteración – paso 2, nodos con frecuencia mínima.



Segunda iteración. Paso 3: se crea un nuevo nodo en la lista de nodos sin símbolo asociado y frecuencia igual a 20.

Ilustración 6. Segunda iteración – paso 3, inserción de un nuevo nodo



Segunda iteración. Paso 4: Aún quedan en la lista de nodos sin procesar los siguientes nodos:

{[a,40],[b,25],[c,15],[Padre(Padre(f,e),d),20]}

Por lo tanto se deben repetir los pasos 2 y 3 nuevamente.

Tercera iteración. Paso 2: se extrae de la lista de nodos sin procesar los dos nodos con frecuencia mínima, en esta iteración los dos nodos con frecuencia mínima corresponden a:

{[c,15],[Padre(Padre(f,e),d),20]}

Tercera iteración. Paso 3: se crea un nuevo nodo en la lista de nodos sin procesar, sin símbolo asociado y frecuencia igual a 35.

Tercera iteración. Paso 4: Aún quedan en la lista de nodos sin procesar los siguientes nodos:

{[a,40],[b,25],[c,15],[Padre(Padre(Padre(f,e),d),c),35]} Por

lo tanto se deben repetir los pasos 2 y 3 nuevamente.

Ilustración 7. Tercera iteración – paso 2, nodos con frecuencia mínima

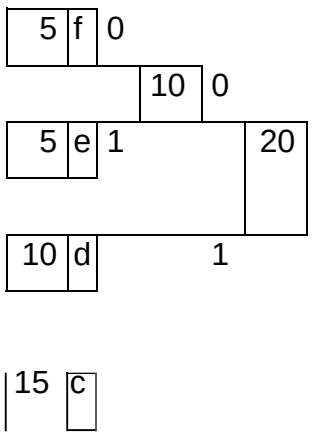
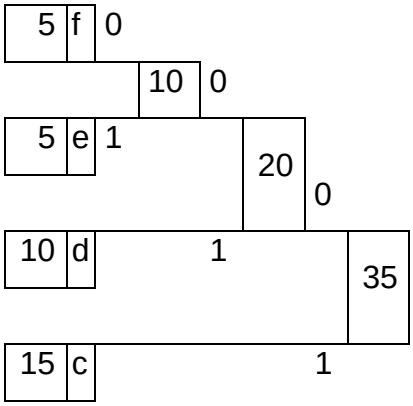


Ilustración 8. Tercera iteración – paso 3, inserción de un nuevo nodo



Cuarta iteración. Pasó 2: se extrae de la lista de nodos sin procesar los dos nodos con frecuencia mínima, en esta iteración los dos nodos con frecuencia mínima corresponden a:

{[b,25],[Padre(Padre(Padre(f,e),d),c),35]}

Cuarta iteración. Paso 3: se crea un nuevo nodo en la lista de nodos sin procesar, sin símbolo asociado y frecuencia igual a 60.

Cuarta iteración. Paso 4: Aún quedan en la lista de nodos sin procesar los siguientes nodos:

{[a,40],[Padre(Padre(Padre(Padre(f,e),d),c),b),60]}

Por lo tanto se deben repetir los pasos 2 y 3 nuevamente.

Ilustración 9. Cuarta iteración – paso 2, nodos con frecuencia mínima

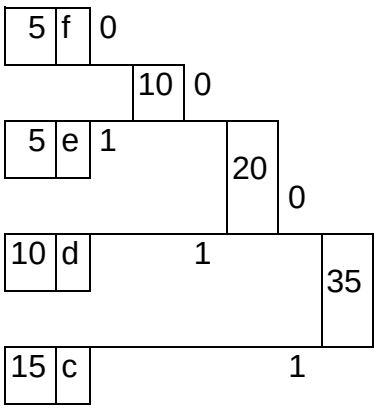
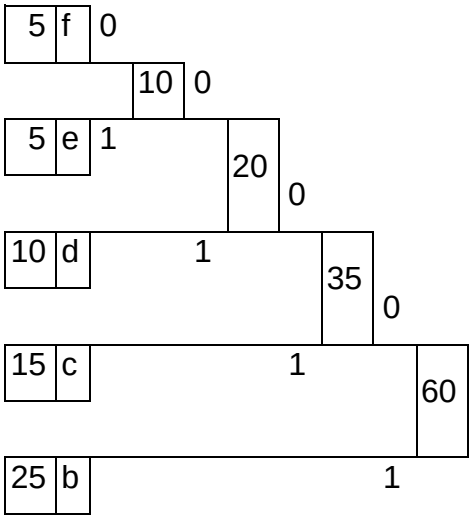


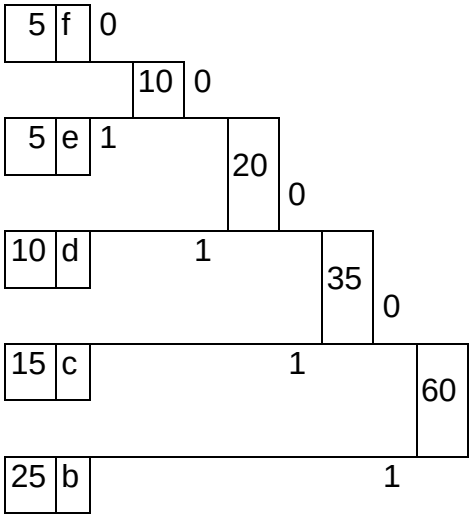
Ilustración 10. Cuarta iteración – inserción de un nuevo nodo



Quinta iteración. Pasó 2: se extrae de la lista de nodos sin procesar los dos nodos con frecuencia mínima, en esta iteración los dos nodos con frecuencia mínima corresponden a:

{[a,40],[Padre(Padre(Padre(Padre(f,e),d),c),b),60]}

Ilustración 11. Quinta iteración – paso 2, nodos con frecuencia mínima

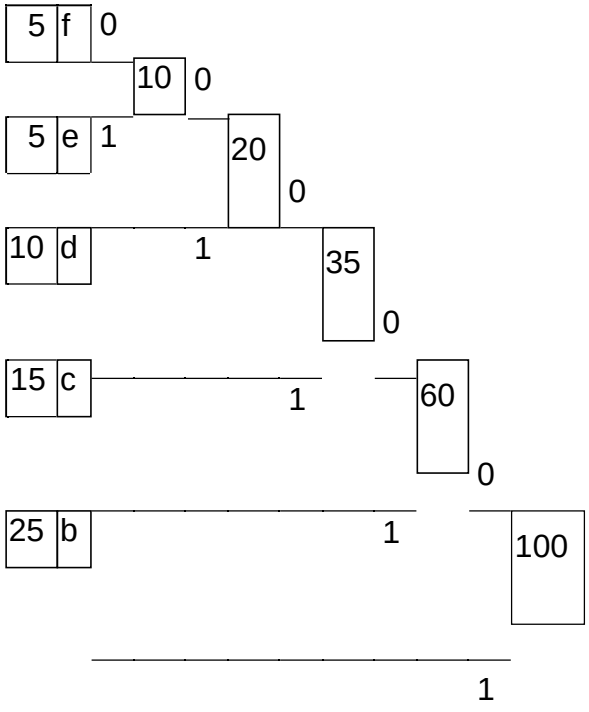


Quinta iteración. Paso 3: se crea un nuevo nodo en la lista de nodos sin procesar, sin símbolo asociado y frecuencia igual a 100.

Quinta iteración. Paso 4: solo existe en la lista de nodos sin procesar un nodo con probabilidad 1, Fin creación del árbol Huffman.

Una vez construido el árbol ya se puede calcular el código correspondiente a cada símbolo del alfabeto. Para calcular los códigos de compresión el árbol se recorre desde la raíz hasta cada uno de los símbolos alojados en las hojas. Para describir el camino recorrido, cada rama del árbol se etiqueta con un número entero. Debido a que cada nodo tiene sólo dos ramas (se trata de un árbol binario) es posible utilizar un único bit. Por ejemplo, las ramas izquierdas se etiquetan con el bit 0 y las derechas con el bit 1. De esta forma, los códigos asignados se muestran en la tabla 11.

Ilustración 12. Quinta iteración – paso 3, inserción de un nuevo nodo



40	a

Tabla 11. Códigos Huffman asignados a cada caracter

Caracter	a	b	c	d	e	f
Código	1	01	001	0001	00001	00000

En la tabla 12 se presenta un ejemplo de codificación usando este código.

Tabla 132. Codificación de una cadena de ejemplo

Cadena	a	b	a	a	e	b	d	f
Código	1	01	1	1	00001	01	0001	00000

Una vez calculados los códigos, el trabajo del codificador consiste en traducir cada símbolo presente en los datos a comprimir, usando el código asociado.

Decodificación Huffman: La decodificación se realiza utilizando el mismo árbol de Huffman generado durante el proceso de codificación, gracias a que los códigos Huffman son instantáneos, la secuencia de códigos se recorre bit a bit y un símbolo es decodificado cada vez que desde la raíz alcanzamos una hojas, se presentara en detalle cada uno de los pasos que se deben llevar acabo para la decodificación de una cadena:

Paso 1: se ubica el puntero de lectura sobre el primer bit (de izquierda a derecha) de la cadena de entrada.
Paso 2: se ubica el puntero al árbol Huffman en el nodo raíz.

Paso 3: se lee el bit al cual apunta el puntero de lectura. Si el bit leído es un cero, el desplazamiento se hace hacia al hijo izquierdo del nodo sobre el cual se esta ubicado, de lo contrario el desplazamiento se hace hacia el hijo derecho. Se avanza el puntero de lectura al siguiente bit en la cadena de entrada.

Paso 4: se examina el tipo de nodo sobre el cual se esta ubicado. Si es una hoja. Se concatena el símbolo asociado a dicha hoja a la cadena de salida. En caso

contrario regresar al paso 3.

Paso 5: si el puntero de lectura aun apunta a un bit de dentro de la cadena de entrada, regresar a el paso 1. En caso contrario la decodificación a finalizado y la cadena de salida es la cadena original.

Se presentara un ejemplo. Utilizando la cadena comprimida en el ejemplo de codificación. Para este ejemplo la cadena codificada es: “101110000101000100000”

Solo se decodificaran los primeros dos símbolos de la cadena, para ilustrar este proceso repetitivo, usando los pasos anteriormente mencionados, la decodificación se realiza de la siguiente forma:

Paso 1: se ubica el puntero de lectura en el primer bit de la cadena de entrada, en este caso un uno.

{[1]01110000101000100000}

Paso 2: se ubica el puntero al árbol Huffman en el nodo raíz.

Paso 3: como el bit al cual apunta el puntero de lectura es uno, el desplazamiento se realiza al hijo derecho del nodo raíz. Se avanza el puntero de lectura un bit hacia la derecha.

{1[0]11100000101000100000}

Paso 4: como se está ubicado en una hoja con símbolo asociado ‘a’, entonces se concatena dicho símbolo a la cadena de salida, que hasta ahora se encontraba vacía.

Cadena codificada=”1[0]11100001010001000000”

Cadena de Salida=”a”

Paso 5: como el puntero de entrada aun apunta a un dentro de la cadena comprimida, se regresa al paso 2.

Segunda iteración. Paso 2: se ubica el puntero al árbol Huffman en el nodo raíz.

Paso 3: como el bit al cual apunta el puntero de lectura es cero, el desplazamiento se realiza hacia el hijo izquierdo del nodo raíz. Se avanza el puntero de lectura un bit hacia la derecha.

{10[1]1100001010001000000}

Segunda iteración. Paso 4: como se está ubicado en un nodo no terminal. Es decir, el nodo donde se está ubicado no es una hoja. Se regresa al paso 3.

Tercera iteración. Paso 3: como el bit al cual apunta el puntero de lectura es tiene valor uno, el desplazamiento se realiza hacia el hijo derecho del nodo actual. Se avanza el puntero de lectura un bit hacia la derecha.

{101[1]100001010001000000}

Tercera iteración. Paso 4: como se está ubicado en una hoja con símbolo asociado ‘b’, entonces se concatena dicho símbolo a la cadena de salida.

Cadena comprimida=”101[1]100001010001000000”

Cadena de Salida=”ab”

Tercera iteración. Paso 5: como el puntero de entrada aun apunta a un dentro de

la cadena comprimida, se regresa al paso 2.

Para decodificar los siguientes seis caracteres se debe seguir el mismo procedimiento, aquí solo se llega hasta este punto, que se considera suficiente para ilustrar el proceso de decodificación.

Este esquema nos permite que si, a la hora de descomprimir, el decodificador Huffman posee el mismo árbol que se hizo para comprimir, la decodificación es tan sencilla como leer bits de la fuente a descomprimir y seguir el camino desde la raíz hacia abajo bifurcando hacia un lado o hacia otro dependiendo del valor del bit. Eventualmente se llega a una hoja, que representa parte o la totalidad del mensaje que se esta decodificando.

Conclusiones y recomendaciones: El principal problema de este algoritmo es que el compresor debe realizar primero un recorrido por los datos (o una porción de los mismos, si se realiza por bloques) a comprimir para reunir las frecuencias (o probabilidades) de los mensajes de la entrada. Como el descompresor no tiene esa posibilidad, ya que sólo recibe los códigos asignados a los mensajes, el árbol ya procesado o las frecuencias de cada mensaje, junto con los datos, deben ser pasados al descompresor. Esto introduce una sobrecarga que hay que evitar de alguna manera. Uno de los programas más antiguos de compresión de datos se llama SQ, de Richard Greenlaw. La solución que aquí se le daba era transmitir el árbol de forma comprimida. Una solución así se encuentra también en uno de los pasos del algoritmo Imploding de PK-ZIP 1.x.

- Compresores aritméticos.

Los compresores aritméticos se valen de un modelo estadístico y un codificador aritmético, de allí su nombre. Al igual que los anteriores la codificación de un símbolo depende de su respectiva frecuencia de aparición.

La codificación aritmética presenta las siguientes características:

- No está restringido a usar una cantidad entera de bits para codificar un

símbolo.

- La cadena de entrada es representada a la salida por un número real R en el rango $0 \leq R < 1$, entre más larga sea la cadena de entrada mayor debe ser la precisión de R.
- Números diferentes pueden representar una misma cadena, siempre que dichos números estén dentro del intervalo al cual está restringido R.

Codificación aritmética. Para construir un código aritmético, se parte de un modelo estadístico que nos proporciona las frecuencias de aparición o las probabilidades de cada símbolo. El proceso de codificación se puede describir como una serie de pasos.

Los siguientes son los pasos para llevar acabo la codificación de una cadena:

Paso 1: se ubica el *Puntero_de_lectura* en el primer símbolo de la *Cadena_de_entrada*. Se toma en la recta numérica el intervalo [INF,SUP), en principio INF es cero y SUP es uno, de esta forma [0,1) es el *Intervalo actual*.

Paso 2: se divide el *Intervalo actual* según la tabla de probabilidades y una *funcion de proporcionalidad*, quedando este dividido en tantos subintervalos como símbolos contenga el alfabeto, el orden el cual se ubican los intervalos de probabilidad en la recta es decreciente. Es decir, el símbolo con mayor probabilidad se localiza en el intervalo [INF,P_j), donde P_j es la probabilidad de dicho símbolo, el siguiente símbolo se localiza en el intervalo [INF+P_j, INF+P_j+P_k), donde $P_j > P_k$ y así sucesivamente hasta que todos los símbolos sean ubicados en el *Intervalo actual*.

Paso 3: se lee el símbolo apuntado por el *Puntero_de_lectura* y se designa como *Simbolo actual*. El nuevo *Intervalo actual* será el subintervalo correspondiente a este símbolo. Por ejemplo, si la localización del símbolo en el *Intervalo actual* es [j,j+P_k), entonces los valores de los límites del nuevo *Intervalo actual* seran INF=j y SUP=j+P_k, donde P_k es la probabilidad asociada al *Simbolo actual*.

Paso 4: se avanza el *Puntero_de_lectura* al siguiente símbolo, si el puntero aun apunta a un símbolo dentro de la *Cadena_de_entrada*, entonces regresar al paso 2. En caso contrario el *Intervalo actual* es el intervalo sobre el que esta definido R y cualquier numero en este intervalo puede representar la *Cadena_de_entrada* codificada.

En el paso 2 del procedimiento se menciona la necesidad de contar con una *función de proporcionalidad*, el objetivo de esta función es asignar a cada símbolo un subintervalo proporcional a su probabilidad dentro del intervalo actual, esto debido a que no siempre es posible asignar a cada símbolo un subintervalo igual a su probabilidad, pues la magnitud del intervalo actual no siempre es uno. Esta función es muy sencilla y en su definición solo se hace uso de una regla de tres simple. Se presentara como se define dicha función:

función_de_proporcionalidad(INF, SUP, P_j){

$Subintevalo_j$ = (INF-SUP)*P_j

RETORNE $Subintevalo_j$
}

La regla de tres simple usada aquí es:

Tabla 13. Regla de tres simple usada en la función de proporcionalidad

MAGNITUD REAL	MAGNITUD PROPORCIONAL
1	INF - SUP
P_j	$SUBINTEVALO_j$

Es claro que ningún símbolo quedara fuera del intervalo [0,1), pues el intervalo ocupado por cada símbolo es equivalente a su probabilidad y sabemos que

$$0 \leq \sum_{i=1}^{n} P_i \leq 1.$$

Para ilustrar el proceso de codificación, se presentara el siguiente ejemplo. Suponiendo

que se desea codificar la cadena “caba”.

Tabla 14. Cadena de entrada, símbolos y probabilidades asociadas

Símbolo	a	b	c
Probabilidad	0.5	0.3	0.2
cadena de entrada			
“caba”			

Para resolver este ejemplo haremos uso del procedimiento descrito anteriormente.

Paso 1: se ubica el puntero de lectura en el primer símbolo de la cadena de entrada.

{[c]aba}

Se definen los limites inferior y superior del intervalo actual como: INF=0,SUP=1.

Paso 2: se divide el intervalo actual según las probabilidades en orden descendente. Es decir entre más alta la probabilidad de un símbolo, el subintervalo asignado estará más cerca del límite inferior del *Intervalo actual*. Para determinar la magnitud del subintervalo se hace uso de la función de proporcionalidad.

Para hallar el subintervalo correspondiente a cada símbolo, se tiene:

$Subintervalo_a=(SUP-INF)*P_a=(1-0)*(0.5)=0.5$

$Subintervalo_b=(SUP-INF)*P_b=(1-0)*(0.3)=0.3$

$Subintervalo_c=(SUP-INF)*P_c=(1-0)*(0.2)=0.2$

Ilustración 13. Codificación aritmética - paso 2, división del intervalo actual

0	1

a [0,0.5)	b[0.5, 0.8)	c[0.8, 1)

Paso 3: se lee el símbolo apuntado por el puntero de lectura, para este ejemplo es ‘c’, se toma el cómo intervalo actual el intervalo [0.8, 1), así INF=0.8 y SUP=1.

Paso 4: se avanza el puntero de lectura:

{c[a]ba}

Como el puntero de lectura aun apunta a un símbolo dentro de la cadena, entonces, se regresa al paso 2.

Segunda iteración. Paso 2: se divide el intervalo actual según las probabilidades en orden descendente. Es decir entre más alta la probabilidad de un símbolo, el subintervalo asignado estará más cerca del límite inferior del *Intervalo actual*. Para determinar la magnitud del subintervalo se hace uso de la función de proporcionalidad.

Para hallar el subintervalo correspondiente a el símbolo ‘a’, se tiene:

$Subintervalo_a=(SUP-INF)*P_a=(1-0.8)*(0.5)=0.1$

$Subintervalo_b=(SUP-INF)*P_b=(1-0.8)*(0.3)=0.06$

$Subintervalo_c=(SUP-INF)*P_c=(1-0.8)*(0.2)=0.04$

Ilustración 14. Codificación aritmética – segunda iteración, paso 2

0.8		1
a [0.8,0.9)	b[0.9, 0.906)	c[0.906, 1)

Segunda iteración. Paso 3: se lee el símbolo apuntado por el puntero de lectura, para este ejemplo es ‘a’, se toma el cómo intervalo actual el intervalo [0.8, 0.9), así

INF=0.8 y SUP=0.9.

Segunda iteración. Paso 4: se avanza el puntero de lectura:

{ca[b]a}

Como el puntero de lectura aun apunta a un símbolo dentro de la cadena de entrada, se regresa al paso 2.

Tercera iteración. Paso 2: se divide el intervalo actual según las probabilidades de los símbolos ubicándolos en orden descendente. Para determinar la magnitud del subintervalo se hace uso de la función de proporcionalidad.

Para hallar el subintervalo correspondiente a cada símbolo, se tiene:

$Subintervalo_a=(SUP-INF)*P_a=(0.8-0.9)*(0.5)=0.05$

$Subintervalo_b=(SUP-INF)*P_b=(0.8-0.9)*(0.3)=0.03$

$Subintervalo_c=(SUP-INF)*P_c=(0.8-0.9)*(0.2)=0.02$

Ilustración 15. Codificación aritmética – tercera iteración, paso 2

0.8		0.9
a [0.8,0.85)	b[0.85,0.88)	c[0.88,0.9)

Tercera iteración. Paso 3: se lee el símbolo apuntado por el puntero de lectura, para este ejemplo es ‘b’, se toma el como intervalo actual el intervalo [0.85, 0.88), así INF=0.85 y SUP=0.88.

Tercera iteración. Paso 4: se avanza el puntero de lectura:

{cab[a]}

Como el puntero de lectura aun apunta a un símbolo dentro de la cadena de entrada, se regresa al paso 2.

Cuarta iteración. Paso 2: se divide el intervalo actual según las probabilidades de los símbolos ubicándolos en orden descendente. Para determinar la magnitud del subintervalo se hace uso de la función de proporcionalidad:

Para hallar el subintervalo correspondiente a cada símbolo, se tiene:

$Subintervalo_a=(SUP-INF)*P_a=(0.88-0.85)*(0.5)=0.015$

$Subintervalo_b=(SUP-INF)*P_b=(0.88-0.85)*(0.3)=0.009$

$Subintervalo_c=(SUP-INF)*P_c=(0.88-0.85)*(0.2)=0.006$

Cuarta iteración. Paso 3: se lee el símbolo apuntado por el puntero de lectura, para este ejemplo es ‘a’, se toma el cómo intervalo actual el intervalo [0.85, 0.865), así INF=0.85 y SUP=0.865.

Ilustración 16. Codificación aritmética – cuarta iteración, paso 2

0.85		0.88
a [0.85,0.865)	b[0.865,0.874)	c[0.874,0.88)

Cuarta iteración. Paso 4: se avanza el puntero de lectura:

{caba[]}

Como el puntero de lectura ahora apunta a un símbolo fuera de la cadena de entrada. El proceso de codificación ha finalizado, el intervalo actual es el intervalo

sobre el que está definido R y cualquier número en este intervalo puede representar la cadena de entrada codificada. Por ejemplo un número que puede representar la cadena de entrada codificada es 0.859375, que escrito en binario es 0.1101110, como el primer dígito siempre es cero no es necesario codificarlo, así se necesitarían solo 7 bits para codificar la cadena “caba”.

Decodificación aritmética. Para llevar a cabo el proceso de decodificación, se deben entregar al descompresor los siguientes datos:

- Un numero R que representa la cadena comprimida.
- El número de símbolos que están codificados en el numero R. Es decir la longitud de la cadena original o de entrada.
- Una lista donde se especifique cada símbolo del alfabeto de la fuente y su respectiva probabilidad.

Después de ver esta lista de parámetros que recibe el descompresor, se aclarara que la cadena “caba” en realidad no se comprimió a 7 bits, pues todos estos parámetros deben ser enviados o almacenados junto con los 7 bits que representan a R.

El proceso de decodificación se realiza de la siguiente forma:

Paso 1: se establece la cadena de salida como una cadena vacía y el contador de símbolos decodificados en cero. Se toma en la recta numérica el intervalo [INF,SUP), en principio INF es cero y SUP es uno, de esta forma [0,1) es el intervalo actual.

Paso 2: se divide el intervalo actual según la tabla de probabilidades y una *funcion de proporcionalidad*, quedando este dividido en tantos subintervalos como símbolos contenga el alfabeto, el orden el cual se ubican los intervalos de probabilidad en la recta es decreciente. Es decir, el símbolo con mayor probabilidad se localiza en el intervalo [INF,P_j), donde P_j es la probabilidad de dicho símbolo, el siguiente símbolo

se localiza en el intervalo [INF+P_j,P_k), donde P_j > P_k y asi sucesivamente hasta que todos los símbolos sean ubicados en la recta.

Paso 3: se ubica el numero R en el intervalo actual, se observa en que subintervalo quedo ubicado R y se concatena el símbolo que corresponde a dicho subintervalo a la cadena de salida. Se toma este subintervalo como el intervalo actual. Por ejemplo, si la localización del símbolo en el intervalo actual es el subintervalo [j,j+P_k), entonces los valores de los límites del nuevo intervalo actual seran INF=j y SUP=j+P_k, donde P_k es una proporción de la probabilidad asociada al símbolo decodificado con respecto a el intervalo actual. Se incrementa el contador de símbolos decodificados.

Paso 4: se evalúa si el contador de símbolos es inferior al número de símbolos representados por el numero R se regresa al paso 2. En caso contrario en la cadena de salida se encuentra decodificada la cadena de símbolos que representa R.

Para ilustrar el procedimiento anterior se presentara un ejemplo. Se va a decodificar la cadena codificada en el ejemplo anterior. En la tabla 15 se muestran los parámetros que son necesarios para realizar la decodificación.

Tabla 15. Para metros pasados al descompresor

Cadena codificada (R)		0.859375	0.1101110
Longitud cadena original			4
Lista – Símbolo/Probabilidad			
Símbolo	A	b	c
Probabilidad	0.5	0.3	0.2

Paso 1: se establece la cadena de salida como una cadena vacía, el contador de símbolos decodificados se hace cero y se toma [0,1) como el intervalo actual. Es decir, INF=0 y SUP=1.

Cadena de salida =""

Contador =0

Intervalo actual =[0, 1)

Paso 2: se divide el intervalo actual según la lista Símbolo/Probabilidad entregada al descompresor y la función de proporcionalidad, ubicando los símbolos de manera decreciente, de la misma forma que se hizo en el proceso de codificación.

Ilustración 17. Descodificación aritmética – paso 2, división del intervalo

0		1
a [0,0.5)	B[0.5, 0.8)	c[0.8, 1)

Paso 3: se ubica R en el intervalo actual.

Se observa que R está ubicado en el subintervalo correspondiente al símbolo 'c'. Se concatena el símbolo 'c' a la cadena de salida y se incrementa en uno el contador. Se toma el subintervalo [INF=0.8, SUP=1) como el intervalo actual.

Ilustración 18. Decodificación aritmética – paso 3

0		1
a [0,0.5)	B[0.5, 0.8)	c[0.8, 1)
		0.859375

Cadena de salida ="c"

Contador =1

Intervalo actual = [0.8, 1)

Paso 4: como el contador de símbolos de codificados es menor que la longitud de la cadena original, se regresa al paso 2.

Segunda Iteración. Paso 2: se divide el intervalo actual según la lista Símbolo/Probabilidad entregada al descompresor y la función de proporcionalidad, ubicando los símbolos de manera decreciente, de la misma forma que se hizo en el proceso de codificación.

Ilustración 19. Descodificación aritmética – segunda iteración, paso 2

0.8		1
a [0.8,0.9)	b[0.9, 0.906)	c[0.906, 1)

Segunda Iteración. Paso 3: se ubica R en el intervalo actual.

Ilustración 20. Decodificación aritmética - segunda iteración, paso 3

0.8		1
a [0.8,0.9)	b[0.9, 0.906)	c[0.906, 1)
0.859375		

Se observa que R está ubicado en el subintervalo correspondiente al símbolo ‘a’. Se concatena el símbolo ‘a’ a la cadena de salida y se incrementa en uno el contador. Se toma el subintervalo [INF=0.8, SUP=0.9) como el intervalo actual.

Cadena de salida =“ca”

Contador =2

Intervalo actual =[0.8, 0.9)

Segunda Iteración. Paso 4: como el contador de símbolos de codificados es menor que la longitud de la cadena original, se regresa al paso 2.

Tercera iteración. Paso 2: se divide el intervalo actual según la lista

Símbolo/Probabilidad entregada al descompresor y la función de proporcionalidad, ubicando los símbolos de manera decreciente, de la misma forma que se hizo en el proceso de codificación.

Ilustración 21. Decodificación aritmética – tercera iteración, paso 2

0.8		0.9
a [0.8,0.85)	b[0.85,0.88)	c[0.88,0.9)

Tercera iteración. Paso 3: se ubica R en el intervalo actual:

Ilustración 22. Decodificación aritmética - paso 3

0.8		0.9
a [0.8,0.85)	b[0.85,0.88)	c[0.88,0.9)
	0.859375	

Se observa que R está ubicado en el subintervalo correspondiente a el símbolo ‘b’. Se concatena el símbolo ‘b’ a la cadena de salida y se incrementa en uno el contador. Se toma el subintervalo [INF=0.85, SUP=0.88) como el intervalo actual.

Cadena de salida =“cab”

Contador =3

Intervalo actual = [0.85, 0.88)

Tercera iteración. Paso 4: como el contador de símbolos de codificados es menor que la longitud de la cadena original, se regresa al paso 2.

Cuarta Iteración. Paso 2: se divide el intervalo actual según la lista Símbolo/Probabilidad entregada al descompresor y la función de proporcionalidad, ubicando los símbolos de manera decreciente, de la misma forma que se hizo en el proceso de codificación.

Ilustración 23. Decodificación aritmética – cuarta iteración, paso 2

0.85		0.88
a [0.85,0.865)	b[0.865,0.874)	c[0.874,0.88)

Cuarta Iteración. Paso 3: se ubica R en el intervalo actual:

Ilustración 24. Decodificación aritmética – cuarta iteración, paso 3

0.85		0.88
a [0.85,0.865)	b[0.865,0.874)	c[0.874,0.88)
0.859375		

Se observa que R está ubicado en el subintervalo correspondiente al símbolo ‘a’. Se concatena el símbolo ‘a’ a la cadena de salida y se incrementa en uno el contador. Se toma el subintervalo [INF=0.85, SUP=0.865) como el intervalo actual.

Cadena de salida =“caba”

Contador =4

Intervalo actual = [0.85, 0.865)

Cuarta Iteración. Paso 4: El contador de símbolos de codificados ya no es menor que la longitud de la cadena original. El proceso de decodificación ha finalizado y la cadena decodificada es Cadena de salida.

Conclusiones y recomendaciones: Este algoritmo es muy eficiente, el problema reside, al igual que los algoritmos anteriores, en que las probabilidades deben ser dadas al recepto; La solución también tiende aquí hacia una modificación "adaptativa con la entrada". Los modelos que van dividiendo los intervalos se convierten ahora en modelos adaptativos y que son capaces de sincronizar con la mínima información a compresor y descompresor. Ejemplos de ellos son el *Dynamic*

Markov Modeling o el *PPMC* (*Partial Predictive Matching*), su explicación está fuera del alcance de esta monografía.

- Compresores Predictivos

Estos compresores procuran predecir el siguiente mensaje de entrada tomando como base de conocimiento los datos procesada hasta ese momento, la codificación predictiva presenta las siguientes características:

Una alta velocidad de compresión/descompresión, al actuar sobre un mensaje cada vez y realizar una predicción que generalmente suele ser de baja complejidad computacional.

- Totalmente adaptativos.
- Implementación generalmente es sencilla.
- Razón de compresión moderada comparada con los anteriores (Semi- Adaptativos).

Codificación predictiva: En este tipo de codificación la compresión está sujeta al éxito o fracaso de una predicción, si el mensaje que se encuentra a la entrada coincide con el predicho, su codificación se podrá hacer con menos bits. Si no, su codificación se hará con más bits.

La codificación predictiva no está representada por un único algoritmo, aun así, se tratara de describir un patrón en los pasos que se deben llevar acabo para codificar una cadena de símbolos usando un algoritmo predictivo, se debe tener un cuenta que estos pasos no son genéricos. Así, los pasos serían los siguientes:

Paso 1: se parte de una base de conocimiento inicial, esta nos ayudara a que las predicciones iniciales tengan una probabilidad de acierto alta.

Paso 2: se hace una predicción acerca del mensaje que llegara. Si acertamos en la predicción, se procede a codificar el mensaje de una forma tal que ocupe menos bits. En caso contrario la codificación se hace más extensa y la cantidad de bits usados aumenta.

Paso 3: se modifica la base de conocimiento con base en el resultado obtenido en el paso 2. Si aún quedan más mensajes por codificar, entonces regresar al paso 2. En caso contrario la codificación ha finalizado.

Para mostrar el funcionamiento de este tipo de codificadores, se presentara un ejemplo:

Para este ejemplo se usara el algoritmo de codificación del compresor llamado "Predictor", que fue inventado en 1987 por Timo Raita y Jukka Teuhola, de la Universidad de Turku, en Finlandia. Fue patentado en 1993 por K. Thomas, y se usa en el *draft* de Internet "PPP Predictor Compression Protocol".
Este algoritmo predice el siguiente carácter a partir de los dos anteriores de la entrada, para ello construye una matriz de 256x256 que guarda en cada casilla m[i] [j] el byte que anteriormente siguió a dos entradas consecutivas de valores ASCII i y j. Cuando se va procesando la entrada, el algoritmo siempre sabe qué dos mensajes precedieron al actual (salvo para los dos primeros mensajes de la entrada, pero esto no es problema). Por ejemplo, si m_1 y m_2 son los dos últimos caracteres recibidos. La predicción para el carácter actual, pongamos m_3, será el carácter contenido en la posición $m[m_1][m_2]$. Si acierta. Es decir, si $m_3 = m[m_1][m_2]$, la salida será solo un bit, colocado a uno (1) informando de que se ha logrado predecir el mensaje actual. Si no acierta, su salida será un bit puesto a cero (0), indicando que no se predijo y a continuación el mensaje no predicho. Además, actualiza la tabla para que la vez siguiente sea capaz de predecir: $m[m_1][m_2] = m_3$.

Se ilustrara con un ejemplo el funcionamiento del “Predictor”. Para desarrollar este ejemplo se tomara como guía la secuencia de pasos descritas anteriormente para la codificación en algoritmos adaptativos.

Ejemplo: se requiere codificar la siguiente la cadena “el sol”.

Paso 1: se parte de una base de conocimiento inicial. Esta base de conocimiento

inicial consiste en una matriz de 256x256. Inicialmente la matriz contiene caracteres que designan un espacio en blanco en cada una de sus posiciones así:

m[i][j]=" " para todo i, j con 0 <= i <256, 0 <= j <256.

La decisión de inicializar la matriz con espacios en blanco en cada uno de sus posiciones no es aleatoria. Si se tiene en cuenta que lo que se va a comprimir es una cadena de texto, el carácter que designa el espacio en blanco es de los que mayor probabilidad de aparición presenta, con lo cual la probabilidad de acierto en los primeros caracteres predichos aumenta.

Paso 2: se hace una predicción acerca del mensaje que llegara. El primer carácter es predicho como un espacio en blanco. Al recibir el primer carácter se presentara que este es "e", por lo tanto la predicción se ha errado. La salida codificada es:

Cadena de salida =0+Bin ("e")

En la cadena de salida Bin ("e") representa el código binario que corresponde al número ASCI del carácter "e".

Paso 3: se modifica la base de conocimiento con base en el resultado obtenido en el paso 2. Cambiamos el contenido de la matriz en la posición m ["a"]["a"], colocando allí el carácter no predicho. Es decir m ["a"]["a"]="e".

Como aún quedan caracteres por codificar se regresa al paso 2.

Segunda iteración. Paso 2: continuando con el siguiente carácter "e", en este caso la predicción también es un carácter en blanco (m["a"]["e"]). Se falla de nuevo y la salida es:

Cadena de salida =0+Bin("e")+0+Bin("l")

Segunda iteración. Paso 3: cambiamos el contenido de la matriz en la posición m[“a”][“e”], colocando allí el carácter no predicho. Es decir m[“a”][“e”]=”l”.

Como aún quedan caracteres por codificar se regresa al paso 2.

Tercera iteración. Paso 2: para el tercer carácter la predicción es m[“e”][“l”], que corresponde con un espacio en blanco. La predicción acierta.

Cadena de salida =0+Bin(“e”)+0+Bin(“l”)+1

Tercera iteración. Paso 3: el contenido de la matriz en la posición m[“e”][“l”] se conserva.

Como aún quedan caracteres por codificar se regresa al paso 2.

Cuarta iteración. Paso 2: se continuara con el cuarto carácter “s”. Para este carácter la predicción es m[“l”][“ ”], que corresponde con un espacio en blanco. La predicción es errada.

Cadena de salida =0+Bin(“e”)+0+Bin(“l”)+1+0+Bin(“s”)

Cuarta iteración. Paso 3: cambiamos el contenido de la matriz en la posición m[“l”][“

”], colocando allí el carácter no predicho. Es decir m[“l”][“ ”]=”s”.

El proceso es repetitivo y se continúa de forma idéntica para los caracteres restantes, al finalizar el proceso se obtiene:

Cadena de salida =0+Bin(“e”)+0+Bin(“l”)+1+0+Bin(“s”)+0+Bin(“o”)+0+Bin(“l”)

El ejemplo anterior solo pretende esbozar el funcionamiento del “Predictor”, debido a que la cadena es muy corta no se puede ver en forma plena su capacidad de compresiones, es lógico que inicialmente las predicciones no sean muy acertadas, pues el algoritmo no ha llenado su base de conocimiento de la cual depende la certeza de sus predicciones.

Este algoritmo es sencillo en su funcionamiento e implementación. Su baja complejidad hace que se pueda considerar como un algoritmo de compresión en tiempo real, como un ejemplo, se dirá que, por ejemplo, el archivo Command.com de la versión MS-DOS 6.20 que ocupa en disco 56.539 bytes queda comprimido a 44.818. Es decir, a aproximadamente un 79% de su tamaño, este resultado es bueno, sobre todo teniendo en cuenta que este es un archivo binario de código, con una distribución que generalmente tiene poca redundancia.

Decodificación predictiva. Como se mencionó al inicio de este capítulo, estos compresores son totalmente adaptativos, siendo así, el compresor no pasa ningún tipo de tabla o encabezado al descompresor, solo se pasan los datos comprimidos, el descompresor también se tiene que aventurar a hacer predicciones, partiendo de los datos comprimidos, la diferencia aquí es que las predicciones buscan rescatar el mensaje original.

Al igual que la codificación, la decodificación predictiva tampoco es única y cada codificador tiene su decodificador particular, a continuación se tratara de hacer un esquema de los pasos a seguir al llevar acabo la decodificación, este algoritmo no es de ninguna forma genérico, pero da una idea general del proceso:

Paso 1: se parte de una base de conocimiento inicial, generalmente igual a la empleada por el codificador.

Paso 2: Basados en los datos de entrada se hacen predicciones que conducirán a la recuperación de los símbolos originales, estas predicciones deben estar sujetas

a las particularidades del codificador.

Paso 3: se modifica la base de conocimiento con base en el resultado obtenido en el paso 2. Si aún quedan más mensajes por decodificar, entonces regresar al paso 2. En caso contrario la decodificación ha finalizado.

Vamos a ilustrar este proceso mediante un ejemplo que parte del ejemplo expuesto en la codificación. El “Predictor” lleva acabo la descompresión de la siguiente forma:

El descompresor parte de la tabla vacía inicial, igual que el compresor, al leer un bit con valor uno (1), sabe que el mensaje que se leído es el que se encuentra en la tabla en la posición indicada por los dos últimos mensajes, si recibe un bit con valor cero (0), sabe que la siguiente información será el mensaje tal cual, y podrá actualizar su tabla al igual que el compresor. Cuando en la secuencia de pasos anteriores se especificó que el descodificador predictivo debe hacer sus predicciones basado en las particularidades del codificador, se hace referencia a que de cualquier modo en la misma cadena de símbolos codificados debe incluir información que le permita al decodificador validar o por lo menos saber en qué momento dado es adecuado hacer una predicción.

Se presentara un ejemplo de descodificación. Se usara como cadena codificada la que obtuvimos como salida en el ejemplo de codificación.

Cadena codificada =0+Bin (“e”)+0+Bin (“l”)+1+0+Bin (“s”)+0+Bin (“o”)+0+Bin (“l”)

Paso 1: se parte de una base de conocimiento inicial, la base de conocimiento inicial para el decodificador del “Predictor”, debe ser la misma base inicial que se usó en el codificador así, la base de conocimiento inicial queda representada por una matriz 256x256, con espacios en blanco en cada una de sus posiciones.

Paso 2: con base en los datos de entrada se hacen predicciones, el decodificador del “Predictor” obtiene el primer bit de la cadena de entrada, como este bit esta

puesto en cero (0), toma los siguientes ocho bits como el carácter no predicho y lo concatena a la cadena de salida.

Cadena de Salida =“e”

Paso 3: se modifica la base de conocimiento con base en el resultado obtenido en el paso 2. Como el iteració decodificado no fue predicho por el codificador, se debe cambiar el contenido de la matriz de conocimiento. Así, se ubica el iteració “e” en m[“a”][“a”](m[“a”][“a”]=”e”).

Como aún hay iteración por decodificar debemos regresar al paso 2.

Segunda iteración. Paso 2: se obtiene de la cadena de entrada el bit que se encuentra inmediatamente después del último carácter decodificado. Como este bit esta puesto en cero (0), toma los siguientes ocho bits como el carácter no predicho y lo concatena a la cadena de salida.

Cadena de Salida =“el”

Segunda iteración. Paso 3: se modifica la base de conocimiento con base en el resultado obtenido en el paso 2. Como el carácter decodificado no fue predicho por el codificador, se debe cambiar el contenido de la matriz de conocimiento. Así, se ubica el carácter “l” en m [“a”][“e”](m[“a”][“e”]=”l”).

Como aún hay caracteres por decodificar debemos regresar al paso 2.

Segunda iteración. Pasó 2: se obtiene de la cadena de entrada el bit que se encuentra inmediatamente después del último carácter decodificado. Como este bit esta puesto en uno (1), debemos buscar en la matriz de conocimiento el caracter predicho. Se concatena a la cadena de salida el símbolo ubicado en m [“e”][“l”], que para este caso el un espacio en blanco.

Cadena de Salida =“el”

Segunda iteración. Paso 3: la iteración decodificado fue predicho por el codificador, no se cambiara el contenido de la matriz de conocimiento.

Como aún hay iteración por decodificar debemos regresar al paso 2.

Tercera iteración. Pasó 2: se obtiene de la cadena de entrada el bit que se encuentra inmediatamente después del último carácter decodificado. Como este bit esta puesto en cero (0), toma los siguientes ocho bits como el carácter no predicho y lo concatena a la cadena de salida.

Cadena de Salida =“el s”

Tercera iteración. Paso 3: se modifica la base de conocimiento con base en el resultado obtenido en el paso 2. Como el carácter decodificado no fue predicho por el codificador, se debe cambiar el contenido de la matriz de conocimiento. Así, se ubica el carácter “s” en m [“l”][“ ”](m[“l”][“ ”]=”s”).

El proceso de descodificación continúa de forma repetitiva hasta que no queden más símbolos en la cadena de entrada.

Conclusiones y recomendaciones:

Predecir eventos futuros no es una tarea fácil en el mundo real, las variables que intervienen en un evento real son excesivas y es prácticamente imposible tenerlas bajo control a todas, debido a esta limitación para realizar predicciones de eventos reales se idealiza en un modelo el comportamiento de casi todas las variables exceptuando las variables de estudio. En el caso de la predicción usada en la compresión de datos las variables no son muchas, es posible tener cero incertidumbres sobre el estado de las mismas y aun así las predicciones no serán un 100% acertada. Debido al comportamiento aleatorio de la fuente, las

predicciones nunca serán infalibles; pero es posible establecer un patrón de comportamiento de la fuente en la medida que interactuamos con ella. Así, las predicciones cambiaran en el tiempo y se amoldaran de forma particular a la fuente, aumentando nuestras probabilidades de acierto.

3.2.2 Compresores basados en diccionario

Estos compresores no requieren un conocimiento de la probabilidad de aparición de cada símbolo, por ello serán útiles en aquellas aplicaciones donde no sea posible conocer las probabilidades de los símbolos, se basan en que algunas cadenas de símbolos se repiten frecuentemente. Las cadenas procesadas hasta el momento pasan a ser parte de un diccionario indexado, el cual puede definirse de forma explícita o implícita, durante el proceso de compresión, si la cadena que se está procesando, coincide con una entrada en el diccionario, la salida emitida por el compresor es el índice que el a dicha cadena; índices que generalmente son códigos de menor longitud que la cadena a la que hace referencia.

Hay dos posibles tipos de diccionarios:

Diccionario estático: Es apropiado si conocemos a priori la estructura repetitiva de los datos. Se almacenan en el diccionario sólo los patrones más frecuentes y sus palabras códigos (suelen ser los índices), si los patrones son pocos, las palabras código son cortas.
Diccionario dinámico: Es apropiado para compresores generales aplicables a fuentes de distintas características. Construyen el diccionario conforme procesan la entrada. En el descompresor se reconstruye el mismo diccionario.

Existen gran variedad de algoritmos basados en el uso de diccionarios, la diferencia fundamental entre todos ellos es la manera en que cada uno crea y gestiona el diccionario. Los esquemas más conocidos LZ77 y LZ78 fueron los propuestos por Abraham Lempel y Jacob Ziv en 1977 y 1978 respectivamente, estos esquemas se han venido optimizando desde el momento de su aparición, en 1982 James Storer y Thomas Szymanski presentaron una optimización del LZ77, la cual reducía el

tamaño de los índices de referencia, esta optimización fue llamada LZSS, en 1984 Terry Welch presento y patento la optimización más conocida del LZ78 y la llamo el LZW, en las secciones siguientes se verá en detalle cada una de estas versiones de los esquemas originales de Abraham Lempel y Jacob Ziv.

- Compresión LZ77

Este compresor tiene sus orígenes en 1977 cuando Abraham Lempel y Jacob Ziv presentaron su modelo de compresión basado en diccionario, hasta la fecha todos los algoritmos de compresión desarrollados eran básicamente compresores estáticos.

Mediante el uso de un diccionario implícito que consiste en una porción de la secuencia previamente codificada. El codificador examina la secuencia de caracteres de entrada a través de una ventana deslizante que consta de dos partes:

- Un buffer de búsqueda: porción de la secuencia previamente codificada.
- Un buffer de adelantamiento: porción de la secuencia no codificada hasta el momento.

Para codificar la secuencia que hay en el buffer de adelantamiento, se mueve un puntero sobre el buffer de búsqueda hasta que se encuentra una secuencia de caracteres que concuerda total o parcialmente con la cadena contenida en el buffer de adelantamiento, posteriormente esta coincidencia se codifica como una tripleta ordenada [*Desplazamiento*, *Longitud*, *Siguiente_simbolo*] la compresión se logra gracias a que la codificación de dicha tripleta ordenado es de menor longitud que la codificación de la secuencia original.

Codificación LZ77, La codificación de una cadena de símbolo se lleva acabo haciendo uso de dos buffers, durante todo el proceso de codificación la cadena de entrada se porciona en pequeñas subcadenas que son almacenadas temporalmente un los buffers mientras se son procesadas una a una, la secuencia que se está codificando es almacenada en el buffer de adelantamiento y la

secuencia de referencia o diccionario implícito es almacenada en el buffer de búsqueda.

Para codificar la secuencia que hay en el buffer de adelantamiento, se mueve un puntero de búsqueda hacia atrás sobre el buffer de búsqueda hasta que encuentra una secuencia de caracteres que coinciden con una secuencia idéntica presente en el buffer de adelantamiento, una vez que ha encontrado la coincidencia de mayor longitud, concatena a la cadena de salida la tripleta ordenada [*Desplazamiento*, *Longitud*, *Siguiente_simbolo*] y se avanza la ventana deslizante en una cantidad igual a la longitud de la coincidencia (*Longitud*), la distancia desde la ubicación del puntero de búsqueda al inicio del buffer de adelantamiento se denomina *Desplazamiento* y *Siguiente_simbolo* es el símbolo contiguo al último símbolo incluido en la cadena coincidente. Si no halla coincidencia se debe concatenar a la cadena de salida el primer símbolo del buffer de adelantamiento y se avanza en un símbolo la ventana deslizante.

La codificación de las tripletas ordenadas [*Desplazamiento*, *Longitud*, *Siguiente_simbolo*], es de suma importancia y determina en gran parte la eficiencia de la compresión, el único requisito que debe cumplir esta codificación es que la tripleta ordenada pueda ser diferenciada de los demás símbolos codificados en la cadena de salida durante el proceso de decodificacion. Una de las formas más eficiente de codificar las tripletas ordenadas es la presentada en 1982 por James Storer y Thomas Szymanski, en su modelo llamado LZSS. La diferencia principal es en la salida, LZ77 siempre da una tripleta ordenada [*Desplazamiento*, *Longitud*, *Siguiente_simbolo*], aún si la coincidencia es de un solo byte. La mejora introducida en el LZSS es el uso de banderas, que ocupan un solo bit y nos informan si lo que viene luego es un símbolo o una tripleta ordenada.

A continuación se describe en forma más detallada el procedimiento de codificación mediante una serie de pasos, debido a la aparición repetitiva varios términos se optara por establecer las siguientes abreviaciones:

Tabla 16. Abreviaciones codificación LZ77

CadEntrada:	Cadena de entrada
CadSalida:	Cadena de salida
BufferBusq:	Buffer de búsqueda
BufferAdel:	Buffer de adelantamiento
PuntBusq:	Puntero de búsqueda, apunta al símbolo que está siendo examinado dentro del buffer de búsqueda.
Puntadel:	Puntero de adelantamiento, apunta al símbolo que debe ser buscado en el buffer de búsqueda.
ContC:	Contador de coincidencias, es usado para llevar la cuenta del número de símbolos consecutivos de una secuencia coincidente con una porción de símbolos en el buffer de adelanto.
MaxC:	Longitud de la coincidencia más significativa hasta el momento.
DSTO:	Desplazamiento, es la distancia que existe entre el primer símbolo en el buffer de adelanto y la coincidencia más significativa hasta el momento, dicha distancia esta medida en cantidad de símbolos.

Paso 1: se establece el tamaño del buffer de Búsqueda y el de Adelanto en M y N respectivamente. Se copian los primeros M símbolos de la cadena de entrada en la cadena de salida y los siguientes N símbolos a partir del símbolo M+1 los copiamos en el buffer de adelanto. Copiamos los primeros M símbolos de la cadena de salida en el buffer de búsqueda.

Paso 2: se ubica el Puntero de Búsqueda en el primer símbolo del buffer de Búsqueda y el puntero de adelanto en el primer símbolo del buffer de adelantamiento, inicializamos las variables de trabajo con los siguientes valores:

ContC=0, MaxC=0, DSTO=0, SigSim = Símbolo actualmente apuntado por PuntBusq.

Paso 3: se comparan los símbolos apuntados por PuntBusq Y PuntAdel. Si ambos símbolos son iguales y ninguno de los dos es igual a *Fin_buffer*, ir al paso 4. De lo

contrario ir al paso 5.

Paso 4: Incrementar en uno ContC, avanzar un símbolo adelante los punteros PuntBusq y PuntAdel. Ir al paso 3.

Paso 5: si ContC es mayor que MaxC, entonces guardar el contenido de ContC en MaxC, en DSTO (Desplazamiento) se guarda la distancia (en símbolos) desde la posición actual del puntero de búsqueda a *fin_buffer*(#) y en SigSim se almacena el carácter al que actualmente apunta PuntAdel.

Si PuntAdel no apunta a *Fin_buffer*, entonces ubicarlo en el primer símbolo. Si PuntBusq no apunta a *Fin_buffer* avanzar PuntBusq en un símbolo.

Paso 6: si PuntBusq o PuntAdel apuntan a *Fin_buffer*(#), entonces ir al paso 7. De lo contrario regresar al paso 3.

Paso 7: si MaxC es mayor que *Longitud_umbral*, entonces Codificar la tripleta ordenada [DSTO, MaxC+1, SigSim], concatenar esta tripleta a la cadena de salida y avanzar la ventana deslizante en una cantidad de símbolos igual a MaxC+1. De lo contrario concatenar a la cadena de salida el primer símbolo del Buffer Adelantamiento y avanzar en un símbolo la ventana deslizante.

Paso 8: si en el buffer de adelantamiento aún hay una cantidad de símbolos mayor que *Longitud_umbral*, entonces ir al paso 2. De lo contrario, los símbolos que haya en el buffer de adelanto se concatenan a la cadena de salida y la codificación habrá finalizado.

Utilizando el anterior procedimiento se va a realizar la codificación de la siguiente cadena:

Cadena de entrada =“XYABCZFABCZEZ”

Paso 1: se usara el mismo tamaño (6) para los dos buffers. El contenido de los buffers es el siguiente:

BufferBusq ="XYABCZ#"

BufferAdel ="FABCZE#"

Cadena de Salida ="XYABCZ"

Paso 2: se ubica el puntero de búsqueda y el de adelanto, en el primer símbolo de su respectivo buffer, así:

Puntbusq ="[X]YABCZ#"

PuntAdel ="[F]ABCZE#"

Inicializamos las siguientes variables de trabajo:

ContC=0, MaxC=0

Paso 3: como los símbolos apuntados por PuntBusq y PuntAdel son diferentes. Saltar al paso 5.

Paso 5: contC no es mayor que MaxC. Se avanza en un símbolo el puntero de búsqueda y se ubica el puntero de adelanto en el primer símbolo.

Puntbusq ="X[Y]ABCZ#"

PuntAdel ="[F]ABCZE#"

Paso 6: ninguno de los punteros apunta a *Fin_buffer*(#), entonces se regresa al paso 3.

Segunda iteración. Paso 3: como los símbolos apuntados por PuntBusq y PuntAdel son diferentes. Saltar al paso 5.

Segunda iteración. Paso 5: conC no es mayor que MaxC. Se avanza en un símbolo el puntero de búsqueda y se ubica el puntero de adelanto en el primer símbolo.

Puntbusq =“XY[A]BCZ#”

PuntAdel =”[F]ABCZE#”

Segunda iteración. Paso 6: ninguno de los punteros apunta a *Fin_buffer*(#), entonces se regresa al paso 3.

Tercera iteración. Paso 3: como los símbolos apuntados por PuntBusq y PuntAdel son diferentes. Saltar al paso 5.

Tercera iteración. Paso 5: conC no es mayor que MaxC. Se avanza en un símbolo el puntero de búsqueda y se ubica el puntero de adelanto en el primer símbolo.

Puntbusq =“XYA[B]CZ#”

PuntAdel =”[F]ABCZE#”

Tercera iteración. Paso 6: ninguno de los punteros apunta a *Fin_buffer*(#), entonces se regresa al paso 3.

Carta iteración. Paso 3: como los símbolos apuntados por PuntBusq y PuntAdel son diferentes. Saltar al paso 5.

Carta iteración. Paso 5: conC no es mayor que MaxC. Se avanza en un símbolo el puntero de búsqueda y se ubica el puntero de adelanto en el primer símbolo.

Puntbusq =“XYAB[C]Z#”

PuntAdel =”[F]ABCZE#”

Cuarta iteración. Paso 6: ninguno de los punteros apunta a *Fin_buffer*(#), entonces se regresa al paso 3.

Quinta iteración. Paso 3: como los símbolos apuntados por PuntBusq y PuntAdel son diferentes. Saltar al paso 5.

Quinta iteración. Paso 5: conC no es mayor que MaxC. Se avanza en un símbolo el puntero de búsqueda y se ubica el puntero de adelanto en el primer símbolo.

Puntbusq =“XYABC[Z]#”

PuntAdel =”[F]ABCZE#”

Quinta iteración. Paso 6: ninguno de los punteros apunta a *Fin_buffer*(#), entonces se regresa al paso 3.

Sexta iteración. Paso 3: como los símbolos apuntados por PuntBusq y PuntAdel son diferentes. Saltar al paso 5.

Sexta iteración. Paso 5: conC no es mayor que MaxC. Se avanza en un símbolo el puntero de búsqueda y se ubica el puntero de adelanto en el primer símbolo.

Puntbusq =“XYABCZ[#]”

PuntAdel =”[F]ABCZE#”

Sexta iteración. Paso 6: El puntero de busqueda apunta a *Fin_buffer*(#).

Sexta iteración. Paso 7: en este caso MaxC es cero, como MaxC es menor que *Longitud*_UMBRAL. Se concatena a la cadena de salida el primer carácter del buffer de adelanto, así:

Cadena de salida =“XYABCZF”

Se avanza la ventana deslizante en un símbolo, de este modo el buffer de búsqueda y el de adelanto ahora contienen:

Bufferbusq =“YABCZF#”

BufferAdel =“ABCZEZ#”

Sexta iteración. Paso 8: como en el buffer de adelanto aún hay símbolos para codificar, entonces regresar al paso 2.

Octava iteración. Paso 2: se ubica el puntero de búsqueda y el de adelanto, en el primer símbolo de su respectivo buffer, así:

Puntbusq =“[Y]ABCZF#”

PuntAdel =“[A]BCZEZ#”

Inicializamos las siguientes variables de trabajo:

ContC=0, MaxC=0

Octava iteración. Paso 3: como los símbolos apuntados por PuntBusq y PuntAdel son diferentes. Saltar al paso 5.

Octava iteración. Paso 5: contC no es mayor que MaxC. Se avanza en un símbolo el puntero de búsqueda y se ubica el puntero de adelanto en el primer símbolo.

Puntbusq =“Y[A]BCZF#”

PuntAdel =“[A]BCZEZ#”

Octava iteración. Paso 6: ninguno de los punteros apunta a *Fin_buffer*(#), entonces se regresa al paso 3.

Novena iteración. Paso 3: como los símbolos apuntados por PuntBusq y PuntAdel

son iguales, se continuara con el paso 4.

Novena iteración. Paso 4: se incrementa en uno a ContC (ContC=1) y se avanza en un símbolo ambos punteros, el de búsqueda y el de adelanto. Ir al paso 3.

Puntbusq =“YA[B]CZF#”

PuntAdel =“A[B]CZEZ#”

Décima iteración. Paso 3: como los símbolos apuntados por PuntBusq y PuntAdel son iguales, se continuara con el paso 4.

Décima iteración. Paso 4: se incrementa en uno a ContC (ContC=2) y se avanza en un símbolo ambos punteros, el de búsqueda y el de adelanto. Ir al paso 3.

Puntbusq =“YAB[C]ZF#”

PuntAdel =“AB[C]ZEZ#”

Undécima iteración. Paso 3: como los símbolos apuntados por PuntBusq y PuntAdel son iguales, se continuara con el paso 4.

Undécima iteración. Paso 4: se incrementa en uno a ContC (ContC=3) y se avanza en un símbolo ambos punteros, el de búsqueda y el de adelanto. Ir al paso 3.

Puntbusq =“YABC[Z]F#”

PuntAdel =“ABC[Z]EZ#”

Duodécima iteración. Paso 3: como los símbolos apuntados por PuntBusq y PuntAdel son iguales, se continuara con el paso 4.

Duodécima iteración. Paso 4: Se incrementa en uno a ContC (ContC=4) y se avanza en un símbolo ambos punteros, el de búsqueda y el de adelanto. Ir al paso

3.

Puntbusq =“YABCZ[F]#”

PuntAdel =“ABCZ[E]Z#”

Decimotercera iteración. Paso 3: como los símbolos apuntados por PuntBusq y PuntAdel son diferentes. Saltar al paso 5.

Decimotercera iteración. Paso 5: contC(4) es mayor que MaxC(0), entonces se guarda en MaxC el contenido de ContC y la distancia en símbolos desde la posición actual de PuntBusq hasta *Fin_buffer*(#) se guarda en DSTO(DSTO=1), en SigSim se almacena el símbolo actualmente apuntado por PuntAdel (SigSim=”E”). Se avanza en un símbolo el puntero de búsqueda y se ubica el puntero de adelanto en el primer símbolo.

Puntbusq =“YABCZF[#]”

PuntAdel =“[A]BCZEZ#”

Decimotercera iteración. Paso 6: PuntBusq apunta a *Fin_buffer*(#), ir al paso 7.

Decimotercera iteración. Paso 7: como MaxC (MaxC=4) es mayor que *Longitud*_UMBRAL(3), se codificara el par ordenado [DSTO, MaxC], así:

Cadena de salida =“XYZABCF[1,4.”E”]”

Se avanza la ventana deslizante en una cantidad de símbolos igual a MaxC, de este modo el buffer de búsqueda y el de adelanto ahora contienen:

Bufferbusq =“FABCZE#”

BufferAdel =“Z#”

Decimotercera iteración. Paso 8: la cantidad de símbolos en el buffer de adelanto es menor que *Longitud*_UMBRAL, por lo tanto se concatena a la cadena de salida los símbolos contenidos en el buffer de adelanto y la codificación habrá finalizado.

Cadena de salida =“XYABCZF[1,4.”E”]Z”

Decodificación LZ77. El proceso de decodificación LZ77 consiste en reemplazar cada una de las tripletas ordenadas [*Desplazamiento*, *Longitud*, *Siguiente_simbolo*] por la secuencia a la cual hace referencia dicha tripleta, el decodificador debe ser capaz de identificar las tripletas ordenadas dentro de la cadena codificada, la identificación e interpretación de las tripletas depende directamente de la codificación aplicada a estas durante el proceso de compresión.

Se presentara los pasos necesarios para llevar acabo la decodificación una secuencia. Los pasos generales son los siguientes:

Paso 1: se ubica el puntero de lectura en el primer símbolo de la cadena codificada.

Paso 2: se lee el símbolo apuntado por el puntero de lectura. Si el símbolo leído no indica el inicio de una tripleta este es concatenado a la cadena de salida, el puntero de lectura se avanza al siguiente símbolo y se continuara en el paso 3. En caso contrario se procede a la interpretación de esta de la siguiente forma:
Paso 2.1: Se retrocede el puntero de lectura en una cantidad de símbolos igual a la suma de los campos *Longitud* y *Desplazamiento*.

Paso 2.2: se lee el símbolo apuntado por el puntero de lectura y lo se concatena a la cadena de salida.

Paso 2.3: Se incrementa el contador de símbolos, si el contador de símbolos es menor que el campo *Longitud* de la tripleta que está siendo interpretada, entonces se regresa al paso 2.2. En caso contrario se ubica el puntero de lectura a apuntar al símbolo contiguo ante a la tripleta inmediatamente interpretada.

Paso 3: si aún no se ha llegado al final de la cadena codificada, entonces regresar al paso 2. En caso contrario la decodificación ha finalizado y la cadena de salida es la cadena original. Es decir la cadena que se tenía antes de aplicar el algoritmo de codificación.

Se presentara un ejemplo donde se ilustra el procedimiento anteriormente descrito, Para este ejemplo se tomara como cadena codificada la cadena de salida del ejemplo presentado en la codificación:

Cadena de entrada =“XYABCZF[1,4.”E”]Z”

Paso 1: se ubica el puntero de lectura en el primer símbolo de la cadena codificada.

Cadena de entrada =“[X]YABCZF[1,4.”E”]Z”

Paso 2: Se lee el símbolo apuntado por el puntero de lectura (“X”). Se concatena a la cadena de salida el símbolo apuntado por el puntero de lectura y se avanza el puntero al siguiente símbolo.

Cadena de Entrada =“X[Y]ABCZF[1,4.”E”]Z”

Cadena de Salida =“X”

Paso 3: Regresar al paso 2.

Segunda iteración. Paso 2: se lee el símbolo apuntado por el puntero de lectura(“Y”). Se concatena a la cadena de salida el símbolo apuntado por el puntero de lectura y se avanza el puntero al siguiente símbolo.

Cadena de Entrada =“XY[A]BCZF[1,4.”E”]Z”

Cadena de Salida =“XY”

Segunda iteración. Paso 3: Regresar al paso 2.

Tercera iteración. Paso 2: se lee el símbolo apuntado por el puntero de lectura(“A”). Se concatena a la cadena de salida el símbolo apuntado por el puntero de lectura y se avanza el puntero al siguiente símbolo.

Cadena de Entrada =“XYA[B]CZF[1,4.”E”]Z”

Cadena de Salida =“XYA”

Tercera iteración. Paso 3: Regresar al paso 2.

Cuarta iteración. Paso 2: se lee el símbolo apuntado por el puntero de lectura(“B”). Se concatena a la cadena de salida el símbolo apuntado por el puntero de lectura y se avanza el puntero al siguiente símbolo.

Cadena de Entrada =“XYAB[C]ZF[1,4.”E”]Z”

Cadena de Salida =“XYAB”

Cuarta iteración. Paso 3: Regresar al paso 2.

Quinta iteración. Paso 2: Se lee el símbolo apuntado por el puntero de lectura(“C”). Se concatena a la cadena de salida el símbolo apuntado por el puntero de lectura y se avanza el puntero al siguiente símbolo.

Cadena de Entrada =“XYABC[Z]F[1,4.”E”]Z”

Cadena de Salida =“XYABC”

Quinta iteración. Paso 3: Regresar al paso 2.

Sexta Iteración. Paso 2: Se lee el símbolo apuntado por el puntero de lectura(“Z”). Se concatena a la cadena de salida el símbolo apuntado por el puntero de lectura y se avanza el puntero al siguiente símbolo.

Cadena de Entrada =“XYABCZ[F][1,4.”E”]Z”

Cadena de Salida =“XYABCZ”

Sexta iteración. Paso 3: Regresar al paso 2.

Séptima Iteración. Paso 2: Se lee el símbolo apuntado por el puntero de lectura (“F”). Se concatena a la cadena de salida el símbolo apuntado por el puntero de lectura y se avanza el puntero al siguiente símbolo.

Cadena de Entrada =“XYABCZF[[ ]1,4.”E”]Z”

Cadena de Salida =“XYABCZF”

Séptima iteración. Paso 3: Regresar al paso 2.

Octava Iteración. Paso 2: El símbolo apuntado por el puntero de lectura indica el incio de una tripleta ordenada[*Desplazamiento*, *Longitud*, *Siguiente_simbolo*]. Entonces se procede a la interpretación de esta de la siguiente forma:

Paso 2.1: Se retrocede el puntero de lectura en cinco símbolos(como lo indica la suma de los campos *Desplazamiento* y *Longitud*) y se establece en cero (0) el contador de símbolos.

Puntero de lectura =“XY[A]BCZF[1,4.”E”]Z”

Cadena de salida =“XYABCZF”

Contador de símbolos =0

Paso 2.2: se lee el símbolo apuntado por el puntero de lectura y lo se concatena a la cadena de salida.

Puntero de lectura =“XY[A]BCZF[1,4.”E”]Z”

Cadena de salida =“XYABCZFA”

Contador de símbolos =0

Paso 2.3: Se incrementa el contador de símbolos y se avanza el puntero de lectura en un símbolo. Se regresa al paso 2.2.

Puntero de lectura =“XYA[B]CZF[1,4.”E”]Z”

Cadena de salida =“XYABCZFA”

Contador de símbolos =1

Segunda Iteración (descodificación tripleta)

Paso 2.2: se lee el símbolo apuntado por el puntero de lectura y lo se concatena a la cadena de salida.

Puntero de lectura =“XYA[B]CZF[1,4.”E”]Z”

Cadena de salida =“XYABCZFAB”

Contador de símbolos =1

Paso 2.3: Se incrementa el contador de símbolos y se avanza el puntero de lectura en un símbolo. Se regresa al paso 2.2.

Puntero de lectura =“XYAB[C]ZF[1,4.”E”]Z”

Cadena de salida =“XYABCZFAB”

Contador de símbolos =2

Tercera Iteración(descodificación tripleta)

Paso 2.2: se lee el símbolo apuntado por el puntero de lectura y lo se concatena a la cadena de salida.

Puntero de lectura =“XYAB[C]ZF[1,4.”E”]Z”

Cadena de salida =“XYABCZFABC”

Contador de símbolos =2

Paso 2.3: Se incrementa el contador de símbolos y se avanza el puntero de lectura en un símbolo. Se regresa al paso 2.2.

Puntero de lectura =“XYABC[Z]F[1,4.”E”]Z”

Cadena de salida =“XYABCZFABC”

Contador de símbolos =3

Cuarta Iteración(descodificación tripleta)

Paso 2.2: se lee el símbolo apuntado por el puntero de lectura y lo se concatena a la cadena de salida.

Puntero de lectura =“XYABC[Z]F[1,4.”E”]Z”

Cadena de salida =“XYABCZFABCZ”

Contador de símbolos =3

Paso 2.3: Se incrementa el contador de símbolos, el contenido del contador de símbolos no es menor que el campo longitud de la tripleta que se está decodificando (*Longitud*=4) se concatena a la cadena de salida el símbolo contenido en el campo *Siguiente_simbolo* de la tripleta que se está decodificando y se ubica el puntero de lectura en el símbolo contiguo a la tripleta decodificada.

Puntero de lectura =“XYABCZF[1,4.”E”][Z]”

Cadena de salida =“XYABCZFABCZE”

Contador de símbolos =4

Octava iteración. Paso 3: Regresar al paso 2.

Novena iteración. Paso 2: Se lee el símbolo apuntado por el puntero de lectura(“Z”). Se concatena a la cadena de salida el símbolo apuntado por el puntero de lectura y se avanza el puntero al siguiente símbolo.

Cadena de Entrada = “XYABCZF[1,4.”E”] [Z]”

Cadena de Salida =“XYABCZFABCZEZ”

Novena iteración. Paso 3: Hemos llegado al final del proceso de descodificación. La cadena decodificada es “XYABCZFABCZEZ” y efectivamente concuerda con la cadena que se usó como entrada en el ejemplo de codificación.

Conclusiones y recomendaciones. Existen distintas variantes mejoradas del algoritmo básico LZ77, cada una de estas mejoras obtienen una mayor eficiencia al optimizar uno de los siguientes aspectos del algoritmo básico, tales aspectos son:

Codificación de tripletas: como se dijo anteriormente, entre menos bits se usen para codificar una tripleta mayor será la compresión obtenida, la optimización en este aspecto se lleva a cabo mediante el uso de bits de bandera como en el caso del LZSS, que usa un bit para identificar si el símbolo leído es el inicio de una tripleta o un literal, de otro lado también se puede optar por una codificación de longitud variable de las tripletas (Pkzip, Zip, LHarc, PNG, gzip, ARJ).

Tamaño de los buffers: el algoritmo básico LZ77 solo hace uso del pasado cercano, es decir el espacio de búsqueda de repeticiones está restringido a un pequeño subconjunto de un total de todos los símbolos procesados con anterioridad, el uso de un buffer de búsqueda limitado tiene su justificación, en aras de un ahorro en el número de bits necesarios para codificar las tripletas, pero hay situaciones en las cuales el uso de un buffer de mayor tamaño aumenta en un alto grado la eficiencia de la compresión, al aumentar el tamaño del buffer de búsqueda las probabilidades de encontrar coincidencias significativas también se incrementan, todo esto con un costo.

- Compresión LZ78

En 1978 Abraham Lempel y Jakob Ziv presentaron un algoritmo de compresión basado en diccionario, este algoritmo a diferencia su anterior trabajo el LZ77 no hace uso de una ventana deslizante, el LZ78 a cambio construye explícitamente un diccionario. La ventaja de usar un diccionario explícito es que elimina la limitación de recurrencia cercana que suponía el LZ77 con su ventana deslizante, al hacer uso de toda la información procesada con anterioridad la probabilidad de hallar secuencias de símbolos repetitivas aumenta.

El diccionario consiste en una tabla con 2 columnas y N filas, la primera columna contiene un índice que se asocia al contenido de la segunda columna, la cual contiene secuencias de símbolos que se han hallado hasta el momento en la cadena de entrada; El proceso de compresión inicia con un diccionario vacío y a medida que la entrada es procesada se colocan en diccionario todas las secuencias que hacen su aparición por primera vez, cuando en la entrada se presenta una

secuencia que ya está en el diccionario, en la cadena de salida esta secuencia es representada por un par ordenado [*índice, Siguiente_símbolo*] cada uno de los campos de este par tiene el siguiente significado:

Índice: especifica la posición que ocupa en el diccionario la secuencia previa más larga coincidente con la entrada.

Siguiente_símbolo: es el código del carácter que viene a continuación de la secuencia coincidente.

Si no se encuentra coincidencia, i = 0, en cualquier caso cada nueva pareja generada [*indice*, *Siguiente_simbolo*] se añade al diccionario. Ahora veremos el algoritmo LZ78 como una secuencia de pasos.

Paso 1: colocar el puntero de lectura en el primer símbolo de la cadena de entrada, asignamos a *Secuencia_en_proceso* una cadena vacía.

Paso 2: Asignamos a *Símbolo_actual* el símbolo apuntado por el puntero de lectura, se avanza el puntero de lectura en un símbolo.

Paso 3: s*ecuencia_en_proceso* + *Símbolo_actual* se halla en el diccionario?

-SI, concatenar *Símbolo_actual* a *Secuencia_en_proceso*.

-NO, entonces concatenar a la *Cadena_de_salida* el par ordenado [índice(*Secuencia_en_proceso*), *Símbolo_actual*], insertar en el diccionario una nueva entrada con el contenido de *Secuencia_en_proceso* + *Símbolo_actual* y asignar a *Secuencia_en_proceso* una cadena vacía.

Paso 4: El puntero de lectura aun apunta a un símbolo dentro de la cadena de entrada?

-SI, entonces regresar al paso 2.

-NO. La cadena *Secuencia_en_proceso* es vacía?

-SI, entonces el proceso de compresión a terminado y en la *Cadena_de_salida* se encuentra cadena comprimida.

-NO, entonces extraer de *Secuencia_en_proceso* el ultimo símbolo y almacenarlo en *Símbolo_actual*, concatenar a la *Cadena_de_salida* un par ordenado [Índice(*Secuencia_en_proceso*), *Símbolo_actual*], el proceso de compresión a terminado y en la *Cadena_de_salida* se encuentra cadena comprimida.

En la secuencia de pasos anteriores, es posible ver como se hace referencia a una función Índice (*parametro*). Se presentara como es el funcionamiento de dicha función:

Función Índice (*parametro*): El parámetro de esta función es una cadena de símbolos, si la cadena *parámetro* se halla en una entrada del diccionario la función retorna el índice correspondiente a dicha entrada, en caso contrario retorna cero.

Para ilustrar este algoritmo de compresión se usara el siguiente ejemplo:

Cadena de entrada =“solosol”

Paso 1: colocar el *Puntero de lectura* en el primer símbolo de la cadena de entrada, asignamos a *Secuencia_en_proceso* una cadena vacía.

Puntero_de_lectura =“[s]olosol”

Secuencia_en_proceso =“”

Paso 2: Asignamos a *Símbolo_actual* el símbolo apuntado por

Puntero_de_lectura, se avanza *Puntero_de_lectura* en un símbolo.

Puntero_de_lectura =“s[o]losol”

Secuencia_en_proceso =“s”

Paso 3: s*ecuencia_en_proceso* + *Símbolo_actual* (“o”) se halla en el diccionario?

-NO, entonces concatenar a *Cadena_de_salida* el par ordenado [Índice (*Secuencia_en_proceso*), *Simbolo_actual*] e insertar en el diccionario una nueva entrada con el contenido de *Secuencia_en_proceso*+*Simbolo_actual*. Asignar a *Secuencia_en_proceso* una cadena vacía.

Paso 4: *Puntero_de_lectura* aun apunta a un símbolo dentro de la cadena de entrada?

-SI, entonces regresar al paso 2.

Tabla 17. Codificación LZ78. Paso 3

Cadena_de_salida	
[0,“s”]	
Secuencia_en_proceso	“”
Puntero_de_lectura	“s[o]losol”
Simbolo_actual	“s”

DICCIONARIO	
Índice	Cadena
1	“s”

Segunda iteración. Paso 2: Asignamos a *Simbolo_actual* el símbolo apuntado por

Puntero_de_lectura, se avanza *Puntero_de_lectura* en un símbolo.

Puntero_de_lectura =“so[l]osol”

Simbolo_actual =“o”

Segunda iteración. Paso 3: s*ecuencia_en_proceso*+*Simbolo_actual* (“o”) se halla en el diccionario?

-NO, entonces concatenar a *Cadena_de_salida* el par ordenado [Índice (*Secuencia_en_proceso*),*Simbolo_actual*] e insertar en el diccionario una nueva entrada con el contenido de *Secuencia_en_proceso*+*Simbolo_actual*. Asignar a *Secuencia_en_proceso* una cadena vacía.

Segunda iteración. Paso 4: *Puntero_de_lectura* aun apunta a un símbolo dentro de la cadena de entrada?

-SI, entonces regresar al paso 2.

Tabla 18. Codificación LZ78. Segunda iteración, paso 3

Cadena_de_salida	
[0,“s”]+[0,“o”]	
Secuencia_en_proceso	“”
Puntero_de_lectura	“so[l]osol”
Simbolo_actual	“o”

DICCIONARIO	
Índice	Cadena
1	“s”
2	“o”

Tercera iteración. Paso 2: Asignamos a *Simbolo_actual* el símbolo apuntado por *Puntero_de_lectura*, se avanza *Puntero_de_lectura* en un símbolo.

Puntero_de_lectura =“sol[o]sol”

Simbolo_actual =“l”

Tercera iteración. Paso 3: s*ecuencia_en_proceso*+*Simbolo_actual* (“l”) se halla en el diccionario?

-NO, entonces concatenar a la *Cadena_de_salida* el par ordenado [Índice(*Secuencia_en_proceso*),*Simbolo_actual*] e insertar en el diccionario una nueva entrada con el contenido de *Secuencia_en_proceso*+*Simbolo_actual*. Asignar a *Secuencia_en_proceso* una cadena vacía.

Tercera iteración. Paso 4: *Puntero_de_lectura* aun apunta a un símbolo dentro de la cadena de entrada?

-SI, entonces regresar al paso 2.

Tabla 19. Codificación LZ78. Tercera iteración, paso 3

Cadena_de_salida	
[0,“s”] +[0,“o”] +[0,“l”]	
Secuencia_en_proceso	“”
Puntero_de_lectura	“sol[o]sol”
Simbolo_actual	“l”

DICCIONARIO	
Índice	Cadena
1	“s”

2	“o”
3	“l”

Cuarta iteración. Paso 2: Asignamos a *Simbolo_actual* el símbolo apuntado por *Puntero_de_lectura*, se avanza *Puntero_de_lectura* en un símbolo.

Puntero_de_lectura =“solo[s]ol”

Simbolo_actual =“o”

Cuarta iteración. Paso 3: s*ecuencia_en_proceso*+*Simbolo_actual* (“o”) se halla en el diccionario?

-SI, concatenar *Simbolo_actual* a *Secuencia_en_proceso*.

Cuarta iteración. Paso 4: *Puntero_de_lectura* aun apunta a un símbolo dentro de la cadena de entrada?

-SI, entonces regresar al paso 2.

Tabla 20. Cuarta iteración. Cuarta iteración, paso 3

Cadena_de_salida	
[0,“s”]+[0,“o”] +[0,“l”] +[0,“e”]	
Secuencia_en_proceso	“o”
Puntero_de_lectura	“solo[s]ol”
Simbolo_actual	“o”

DICCIONARIO	
Indice	Cadena
1	“s”
2	“o”

3	“l”

Quinta iteración. Paso 2: Asignamos a *Simbolo_actual* el símbolo apuntado *Puntero_de_lectura*, se avanza *Puntero_de_lectura* en un símbolo.

Puntero_de_lectura =“solos[o]l”

Simbolo_actual =“s”

Quinta iteración. Paso 3: s*ecuencia_en_proceso*+*Simbolo_actual* (“os”) se halla en el diccionario?

-SI

Quinta iteración. Paso 4: *Puntero_de_lectura* aun apunta a un símbolo dentro de la cadena de entrada?

-SI, entonces regresar al paso 2.

Tabla 21. Codificación LZ78. Quinta iteración, paso 3

Cadena_de_salida	
[0,“s”]+[0,“o”] +[0,“l”] +[2,“s”]	
Secuencia_en_proceso	“”
Puntero_de_lectura	“solos[o]l”
Simbolo_actual	“s”

DICCIONARIO	
Índice	Cadena
1	“s”
2	“o”

3	“l”
4	“os”

Sexta iteración. Paso 2: Asignamos a *Simbolo_actual* el símbolo apuntado por el puntero de lectura, se avanza el puntero de lectura en un símbolo.

Puntero_de_lectura =“soloso[l]”

Simbolo_actual =“o”

Sexta iteración. Paso 3: s*ecuencia_en_proceso*+*Simbolo_actual* (“o”) se halla en el diccionario?

-SI, concatenar “o” a *Secuencia_en_proceso*.

Sexta iteración. Paso 4: *Puntero_de_lectura* aun apunta a un símbolo dentro de la cadena de entrada?

-SI, entonces regresar al paso 2.

Tabla 22. Codificación LZ78. Sexta iteración, paso 3

Cadena_de_salida	
[0,“s”]+[0,“o”] +[0,“l”] +[2,“s”]	
Secuencia_en_proceso	“o”
Puntero_de_lectura	“soloso[l]”
Simbolo_actual	“o”

DICCIONARIO	
Índice	Cadena
1	“s”

2	“o”
3	“l”
4	“os”

Séptima iteración. Paso 2: Asignamos a *Simbolo_actual* el símbolo apuntado por *Puntero_de_lectura*, se avanza *Puntero_de_lectura* en un símbolo.

Puntero_de_lectura =“solosol[]”

Simbolo_actual =“l”

Séptima iteración. Paso 3: s*ecuencia_en_proceso*+*Simbolo_actual* (“ol”) se halla en el diccionario?

-SI, concatenar “l” a *Secuencia_en_proceso*.

Séptima iteración. Paso 4: *Puntero_de_lectura* ya no apunta a ningún carácter dentro de la cadena de entrada. Se examina la cadena *Secuencia_en_proceso* y notamos que no esta vacia, por lo tanto se extrae de *Secuencia_en_proceso* el ultimo símbolo y lo almacenamos en *Simbolo_actual*, concatenar a la c*adena_de_salida* un par ordeando [Índice(“o”),”l”], el proceso de compresión a terminado y en *Cadena_de_salida* se encuentra cadena comprimida.

Cadena_de_salida =[0,“s”]+[0,“o”] +[0,“l”] +[2,“s”]+[2,”L”]

Tabla 23. Codificación LZ78. Séptima iteración, paso 3

Cadena_de_salida	
[0,“s”]+[0,“o”] +[0,“l”] +[2,“s”]	
Secuencia_en_proceso	“ol”
Puntero_de_lectura	“solosol”[]
Simbolo_actual	“l”

DICCIONARIO	
Índice	Cadena
1	“s”
2	“o”
3	“l”
4	“os”

Decodificación LZ78. El proceso de decodificación es muy similar al de codificación, el descompresor va construyendo un diccionario idéntico al realizado por el compresor, cuando se detecta un par ordenado [*índice*, SÍMBOLO] el descompresor concatena a la cadena de salida la entrada que corresponde en el diccionario al índice citado.

Se presentara como sería el proceso de descompresión descrito como una secuencia de pasos:

Paso 1: se ubica el puntero de lectura en el primer par ordenado de la cadena de entrada.

Paso 2: *Puntero_de_lectura* apunta a un par ordenado [*índice*,SIMBOLO]?

-SI:

Concatenar a la cadena de salida el contenido de la entrada que corresponde al *índice*.

Concatenamos SÍMBOLO a la cadena referenciada por *índice*(Cadena(*índice*)) y agregamos el resultado de dicha concatenación al diccionario.

Se avanza el puntero de lectura al siguiente par ordenado y se ejecuta nuevamente el paso 2.

-NO:

Fin descompresión, la cadena de salida es la cadena descomprimida.

Se presentara un ejemplo de descodificación siguiendo la secuencia de pasos anteriores. Para este ejemplo se tomara como cadena de entrada la cadena de codificada en el apartado anterior.

Cadena_de_entrada =[0,“s”]+[0,“o”] +[0,“l”] +[2,“s”]+[2,”L”]

Paso 1: se ubica *Puntero_de_lectura* en el primer par ordenado de *Cadena_de_entrada*.

Puntero_de_lectura =[[0,“s”]]+[0,“o”]+[0,“l”]+[2,“s”]+[2,”L”]

Paso 2: *Puntero_de_lectura* apunta a un par ordenado?

-SI.

El valor de *índice* es cero (0), por tanto el par no se corresponde con ninguna entrada del diccionario, siendo así la cadena referenciada por índice es una cadena vacía.

Concatenamos SIMBOLO (“s”) a la cadena referenciada por *índice* (“”), el resultado de la concatenación lo agregamos como una nueva entrada en el diccionario y también se lo se concatena a *Cadena_de_salida*.

Se avanza *Puntero_de_lectura* al siguiente par ordenado.

Después de aplicar el paso 2, el estado de nuestras variables es:

Tabla 24. Decodificación LZ78. Paso 2

Puntero_de_lectura	[0,“s”]+[[0,“o”]]+[0,“l”]+[2,“s”]+[2,”L”]
SIMBOLO	“s”
Índice	0
Cadena(*índice*)	“”
Cadena_de_salida	“s”
DICCIONARIO	
Índice	Cadena
1	“s”

Se ejecuta nuevamente el paso 2.

Segunda iteración. Paso 2: *Puntero_de_lectura* apunta a un par ordenado?

-SI.

El valor de *índice* es cero (0), por tanto el par no corresponde con ninguna entrada del diccionario, siendo así la cadena referenciada por índice una cadena vacía.

Concatenamos SIMBOLO (“o”) a la cadena referenciada por *índice* (“”), el resultado de la concatenación lo agregamos como una nueva entrada en el diccionario y también se lo se concatena a *Cadena_de_salida*.

Se avanza *Puntero_de_lectura* al siguiente par ordenado.

Después de aplicar el paso 2, el estado de nuestras variables es:

Tabla 25. Decodificación LZ78. Segunda iteración, paso 2

Puntero_de_lectura	[0,“s”]+[0,“o”]+[[0,“l”]]+[2,“s”]+[2,”L”]
SIMBOLO	“o”
índice	0
Cadena(*índice*)	“”

Cadena_de_salida	“so”
DICCIONARIO	
Índice	Cadena
1	“s”
2	“o”

Se ejecuta nuevamente el paso 2.

Tercera iteración. Paso 2: *Puntero_de_lectura* apunta a un par ordenado?

-SI.

El valor de *índice* es cero (0), por tanto no se corresponde con ninguna entrada del diccionario, siendo así la cadena referenciada por índice es una cadena vacía.

Concatenamos SIMBOLO (“l”) a la cadena referenciada por *índice* (“”), el resultado de la concatenación lo agregamos como una nueva entrada en el diccionario y también se lo se concatena a *Cadena_de_salida*. Se avanza *Puntero_de_lectura* al siguiente par ordenado. Después de aplicar el paso 2, el estado de nuestras variables es: Tabla 26. Decodificación LZ78. Tercera iteración, paso 2

Puntero_de_lectura	[0,“s”]+[0,“o”]+[0,“l”]+[[2,“s”]]+[2,”L”]
SIMBOLO	“l”
índice	0
Cadena(*índice*)	“”
Cadena_de_salida	“sol”
DICCIONARIO	
Índice	Cadena
1	“s”
2	“o”
3	“l”

Se ejecuta nuevamente el paso 2.

Cuarta iteración. Paso 2: *Puntero_de_lectura* apunta a un par ordenado?

-SI.

El valor de *índice* es dos (2), este indice se corresponde con la cadena “o”.

Concatenamos SIMBOLO (“s”) a la cadena referenciada por *índice* (“o”), el resultado de la concatenacion(“os”) lo agregamos como una nueva entrada en el diccionario y también se lo se concatena a *Cadena_de_salida*.

Se avanza *Puntero_de_lectura* al siguiente par ordenado.

Después de aplicar el paso 2, el estado de nuestras variables es:

Tabla 27. Decodificación LZ78. Cuarta iteración, paso 2

Puntero_de_lectura	[0,“s”]+[0,“o”]+[0,“l”]+[2,“s”]+[[2,”l”]]
SIMBOLO	“s”
índice	2
Cadena(*índice*)	“o”
Cadena_de_salida	“solos”
DICCIONARIO	
Índice	Cadena
1	“s”
2	“o”
3	“l”
4	“os”

Se ejecuta nuevamente el paso 2.

Quinta iteración. Paso 2: *Puntero_de_lectura* apunta a un par ordenado?

-SI.

El valor de *índice* es dos (2), este indice se corresponde con la cadena “o”.

Concatenamos SIMBOLO (“l”) a la cadena referenciada por *índice* (“o”), el resultado de la concatenacion(“ol”) lo agregamos como una nueva entrada en el diccionario y también se lo se concatena a *Cadena_de_salida*.

Se avanza *Puntero_de_lectura* al siguiente par ordenado.

Después de aplicar el paso 2, el estado de nuestras variables es:

Tabla 28. Decodificación LZ78. Quinta iteración, paso 2

Puntero_de_lectura	[0,“s”]+[0,“o”]+[0,“l”]+[2,“s”]+[2,”l”][]
SIMBOLO	“s”
índice	2
Cadena(*índice*)	“o”
Cadena_de_salida	“solosol”
DICCIONARIO	
Indice	Cadena
1	“s”
2	“o”
3	“l”
4	“os”

Se ejecuta nuevamente el paso 2.

Sexta iteración. Paso 2: *Puntero_de_lectura* ya no apunta a un par ordena dentro de *Cadena_de_entrada*, por lo tanto el proceso de decodificación ha finalizado y en *Cadena_de_salida* se encuentra la cadena decodificada.

Cadena_de_salida =“solosol”

Con el ejemplo anterior es claro que el proceso de descodificación es bastante sencillo, la complejidad de dicho algoritmo radica en la manera de identificar el inicio y el fin de un par ordenado.

Conclusiones y recomendaciones. Es un poco más rápido que el LZ77 debido a que hace menos comparaciones.

3.2.3 Compresión mediante transformada

Esta técnica de compresión consiste en expresar los datos que se pretenden comprimir en una forma que proporcione algún beneficio, bien sea que aplicando la transformada el volumen de los datos se reduzca o que al aplicar la transformada la redundancia presente en los datos pueda ser explotada con más eficiencia por cualquier otra técnica o algoritmo de compresión.

Las transformadas que se trataran en este apartado son transformadas sin pérdida o reversibles que generalmente consisten en una reordenación de los datos, obteniendo como salida un volumen de datos igual al original; Pero con la diferencia de que los datos de salida presentan una alta redundancia.

- Transformada de Borrows-Weller (BWT).

El 10 de Mayo del año 1994, Michael Burrows y David Wheeler publicaron un reporte de investigación titulado, "A Block-sorting Lossless Data Compression

Algorithm", allí se describe un algoritmo para la compresión de datos que se desarrolló tomando como base una transformación descubierta por Wheeler en 1983 y que hasta ese momento no había sido publicada.

Básicamente la BWT es un algoritmo que toma un bloque de datos y lo reorganiza usando un algoritmo de ordenación, el resultado es un bloque que contiene exactamente los mismos elementos que el bloque original, la única diferencia es la forma en la que están ordenados, la transformación es reversible, por consiguiente es posible recuperar el orden original de los elementos sin ninguna distorsión, es decir sin pérdida alguna de los datos originales.

Transformada Borrows-Weller. El procedimiento de transformación de un bloque de datos mediante la BWT descrito como una secuencia de pasos.

Paso 1: considerar una cadena de entrada de longitud n, denominada Cadena [0].

Paso 2: aplicar una rotación a la izquierda a la cadena para obtener la consecuente *Matriz de rotación*.

Cadena[1]=RotarIzq(Cadena[0])

Cadena[2]=RotarIzq(Cadena[1])

Cadena[3]=RotarIzq(Cadena[2])

...

Cadena[n-1]=RotarIzq(Cadena[n-2])

Paso 3: ordenar las cadena Cadena[0],Cadena[1], Cadena[2], …, Cadena[n-1] en orden alfabético decreciente.

Paso 4: la salida que genera la BWT tiene dos componentes. La primera

componente es la última “columna” de presente en la matriz de cadenas ordenadas (la columna denominada L), y la segunda componente es un índice (i), el valor de este indice corresponde con la posición que ocupa Cadena[1] dentro de la matriz de cadenas ordenadas.

La ilustración del procedimiento anterior, se hará por medio el ejemplo expuesto a continuación:

Cadena_de_entrada =“pocococococompro”

Longitud Cadena =15

Paso 1: hacemos Cadena[0] igual al contenido de *Cadena_de_entrada*.

Paso 2:

Tabla 29. Transformada BWT. *Matriz rotación,* paso 2

Cadena[0]	p	o	c	o	c	o	c	o	c	o	m	p	r	o
Cadena[1]	o	c	o	c	o	c	o	c	o	m	p	r	o	p
Cadena[2]	c	o	c	o	c	o	c	o	m	p	r	o	p	o
Cadena[3]	o	c	o	c	o	c	o	m	p	r	o	p	o	c
Cadena[4]	c	o	c	o	c	o	m	p	r	o	p	o	c	o
Cadena[5]	o	c	o	c	o	m	p	r	o	p	o	c	o	c
Cadena[6]	c	o	c	o	m	p	r	o	p	o	c	o	c	o
Cadena[7]	o	c	o	m	p	r	o	p	o	c	o	c	o	c
Cadena[8]	c	o	m	p	r	o	p	o	c	o	c	o	c	o
Cadena[9]	o	m	p	r	o	p	o	c	o	c	o	c	o	c
Cadena[10]	m	p	r	o	p	o	c	o	c	o	c	o	c	o
Cadena[11]	p	r	o	p	o	c	o	c	o	c	o	c	o	m
Cadena[12]	r	o	p	o	c	o	c	o	c	o	c	o	m	p
Cadena[13]	o	p	o	c	o	c	o	c	o	c	o	m	p	r

Paso 3: ver tabla 30.

Paso 4: consideramos L = “ooooopcccromp” y el índice i=5, donde i es indica la posición de la Cadena[1] dentro de la matriz rotada y ordenada.

La salida de la transformada BWT se puede expresar como el un par ordenado (L,i), para este ejemplo la salida es la siguiente:

L= “ooooopcccromp”

i=5

Es claro que si aplicamos a L uno algoritmo de los algoritmos de compresión vistos anteriormente, por ejemplo si usamos RLE, se tiene que:

L= “5op4cromp”

i=5

Al final hemos logrado comprimir la cadena “pocococococompro” que tiene una longitud de 14 símbolos, ahora la información compresa esta representada por el par ordenado [50p4cromp,6].

Tabla 30. Transformada BWT. Matriz ordenada, paso 3

Cadena[2]	c	o	c	o	c	o	c	o	m	p	r	o	p	o
Cadena[4]	c	o	c	o	c	o	m	p	r	o	p	o	c	o
Cadena[6]	c	o	c	o	m	p	r	o	p	o	c	o	c	o
Cadena[8]	c	o	m	p	r	o	p	o	c	o	c	o	c	o
Cadena[10]	m	p	r	o	p	o	c	o	c	o	c	o	c	o
Cadena[1]	o	c	o	c	o	c	o	c	o	m	p	r	o	p
Cadena[3]	o	c	o	c	o	c	o	m	p	r	o	p	o	c

Cadena[5]	o	c	o	c	o	m	p	r	o	p	o	c	o	c
Cadena[7]	o	c	o	m	p	r	o	p	o	c	o	c	o	c
Cadena[9]	o	m	p	r	o	p	o	c	o	c	o	c	o	c
Cadena[13]	o	p	o	c	o	c	o	c	o	c	o	m	p	r
Cadena[0]	p	o	c	o	c	o	c	o	c	o	m	p	r	o
Cadena[11]	p	r	o	p	o	c	o	c	o	c	o	c	o	m
Cadena[12]	r	o	p	o	c	o	c	o	c	o	c	o	m	p

Transformada borrows-weller Inversa. Una vez se tiene un par ordenado [L,Indice], es de esperarse que por la reversibilidad de la BWT podamos recuperar la cadena original, para lograr este propósito es preciso seguir los siguientes pasos:

Paso 1: considerar la cadena L de longitud n y el índice i como entrada.

Paso 2: Formar la columna F ordenando los elementos de L en orden alfabético decreciente.

Paso 3: construir el arreglo T de acuerdo con el siguiente esquema:

T[i] es el lugar que ocupa el i-ésimo elemento de F en la columna L.

Paso 4: Producir la salida siguiendo el procedimiento:

k =i

DESDE j=0 HASTA n

{ Salida[j] = L[k]

K = T[k]

}

Paso 5: considerar Salida como la cadena decodificada.

Ejemplo aplicación del algoritmo BWT Inversa. Para este ejemplo se decodificara la cadena codificada en el ejemplo de codificación BWT.

Aplicando los pasos descritos anteriormente se tiene:

Paso 1: se reciben como cadena de entrada a L="ooooopcccromp" y i=5

Paso 2: se obtiene la tabla 31.

Paso 3: El elemento F[1] (la primera 'c') se encuentra en la posición 7 en L, por tanto T[1]=7. F[1] (siguiente 'c') la encontramos en la posición 6 de L por tanto T[1]=8. F[2] (siguiente 'c') está en la posición 9 por tanto T[2]=9 y así sucesivamente hasta llegar a F[14] (el carácter 'r') donde T[14]=11 porque en esa posición está ubicado 'r' en L. Con ello se obtiene:

Tabla 31. Transformada BWT Inversa. Paso 2

L	o	o	o	o	o	p	c	c	c	c	r	o	m	p
F	c	c	c	c	m	o	o	o	o	o	o	p	p	r
Pos	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Tabla 32. Transformada BWT Inversa. Paso 3

T	7	8	9	10	13	1	2	3	4	5	12	6	14	11
Pos	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Paso 4: Empezamos con un valor inicial de k=6 que corresponde al índice de entrada, (i=6) por lo que el primer carácter de salida es S[0]=L[6] o el carácter 'p'. k toma entonces el valor de k = T[6] = 1 y entonces S[1]=L[1] que corresponde con 'o'.

Para k=i=6

Al finalizar el proceso anterior se presentara que en la columna rotulada como L[k] está contenida la cadena original. Esto muestra la reversibilidad de la BWT.

Paso 5: La cadena de salida es “pococococompro”.

Conclusiones y recomendaciones. Al usar esta transformada sobre textos en inglés existen palabras, muy comunes de la forma como what o that, donde al escogerse las “sartas” ordenadas que iniciaran con h, habría una secuencia muy grande y repetitiva de t y w (susceptibles a comprimirse). En el español también sucede esto.

Tabla 3. Transformada BWT Inversa. Paso 4

Salida		L[k]
S[0]=	L[6]=	‘p’
S[1]=	L[T[6]]=L[1]	‘o’
S[2]=	L[T[1]]=L[7]	‘c’
S[3]=	L[T[7]]=L[2]	‘o’
S[4]=	L[T[2]]=L[8]	‘c’
S[5]=	L[T[8]]=L[3]	‘o’
S[6]=	L[T[3]]=L[9]	‘c’
S[7]=	L[T[9]]=L[4]	‘o’
S[8]=	L[T[4]]=L[10]	‘c’
S[9]=	L[T[10]]=L[5]	‘o’
S[10]=	L[T[5]]=L[13]	‘m’
S[11]=	L[T[13]]=L[14]	‘p’
S[12]=	L[T[14]]=L[11]	‘r’
S[13]=	L[T[11]]=L[12]	‘o’

Como se puede deducir fácilmente el poder de compresión de la transformada estará unido a las propiedades probabilísticas del idioma (en el caso de textos) pues en definitiva la transformada es solo una forma para hacer explícita la redundancia presente en el mismo.

3.2.4 Compresión de texto

En este capítulo hablaremos de la compresión de texto, este habla básicamente de formas alternativas de representación que permiten almacenar las misma información en un menor espacio, estos están dentro de la compresión de datos sin pérdida de información ya que el mensaje que es codificado o comprimido coincide con el mensaje original enviado, los fundamentos de estas técnicas radica en que permita que en el identificar las regularidades con las que se construye un mensaje en lenguaje natural y, a partir de ellas, eliminar la redundancia subyacente a la información contenida en el mensaje original.

Esto nos permite reducir el espacio que ocupado en nuestros dispositivos con el mensaje comprimido, aumentando la capacidad de almacenamiento de nuestros dispositivos, no solo esto nos ofrece este tipo de compresión de texto sino que a la hora de transferencia de datos nos ayuda a que sean más eficientes y rápidas, también esto nos permite que nosotros podamos pre visualizar los textos que están dentro de la compresión para así verificar su contenido y no llevar a cabo su descompresión, como muestra de su eficiencia en los procesos de búsqueda de información podemos hacer una comparación donde los datos de texto comprimidos pueden ser 8 veces más eficientes que en el texto original.

Estas compresiones de texto pueden marcan poca diferencia en las tecnologías que presenta una buena velocidad de computo respecto a los procesadores respecto a la velocidad de trasferencia de disco, aunque para las grandes tecnológicas no podría significar un cambio muy drástico, si lo es para las pequeñas tecnológicas las cuales si podría mostrar un cambio positivos, los cuales puede ayudar a algunos contextos de aplicación reduciendo el coste del almacenamiento.

Algunos de las codificaciones de la compresión de texto, son:

Codificación de redundancia mínima:

Estos son códigos libres de prefijo, donde la longitud media de palabra de código

llega a ver óptima para un conjunto de probabilidades, estos nos indica que la longitud de una palabra de un código es asignada cada símbolo, lo cual deberá ser inversamente proporcional a su probabilidad en el mensaje, en este se plantea por Shannon –Fano y Huffman, estos plantean una construcción de un árbol de códigos en que los símbolos más probables aparecen próximo a la raíz, mientras que los símbolos menos probables aparecen más lejos de la raíz, en este árbol tendrá profundidad donde aparecen algunos cerca a la raíz y otros más lejos de la raíz, en este árbol los símbolos crece a medida que decrece la probabilidad de cada uno de los símbolos.

Codificación aritmética: Está basada en una frecuencia de aparición de símbolos en el mensaje, pero al tener en cuenta este fundamento difiere del anterior dado que la codificación aritmética considera la posibilidad de representar los símbolos con un número fraccional de bits.

Códigos de enteros: Estos plantean una buena solución para todos estos casos en los que el tamaño del alfabeto fuente no puede ser estimado apriori. De forma genérica, los códigos de enteros mapean un número en una secuencia de bits de longitud variable que se utiliza para la representación de dicho número.

3.3 COMPRESIÓN CON PERDIDA

3.3.1 Compresión de imágenes

En un principio la compresión de imágenes se limitaba a comprimir el ancho de banda en las transmisiones de vídeo mediante métodos analógicos, con la llegada de las computadoras digitales, los métodos de compresión analógicos fueron dejando paso a la compresión digital, el hecho de que en la actualidad se hayan adoptado diversos estándares internacionales, ha hecho que este campo haya avanzado de manera considerable.

A medida que la Informática ha avanzado, las imágenes se han convertido en una pieza muy importante de ésta, actualmente surgen cada día más entornos gráficos orientados a múltiples aplicaciones. En los comienzos la representación de los datos en las computadoras se hacía por medio de texto, el código ASCII, la

compresión de este tipo de datos es sencilla y además está limitada a una compresión sin perdida, pues es inaceptable la perdida de datos en un archivo de texto o binario.

Debido al amplio uso que se le da a las imágenes en el campo de la informática, se hace necesario reducir la cantidad de datos necesarios para representarlas digitalmente, las técnicas de compresión en las imágenes al igual que las técnicas vistas anteriormente también se basan en la eliminación de datos redundantes, expresado esto matemáticamente, equivale a transformar una distribución bidimensional de pixels en un conjunto de datos estadísticos sin correlacionar, esta transformación (compresión) es aplicada a las imágenes antes de que sean almacenadas.

Se presentara ahora cuales son los criterios sobre los cuales se basan estas técnicas de compresión para conseguir la reducción deseada de los datos que representan las imágenes. Estos criterios son:

a. Reducir matices de color en la imagen

b. Reducir la resolución de color respecto a la intensidad de luz prevaleciente

c. Reducir partes pequeñas, invisibles de la imagen

Todas estas técnicas están basadas en un conocimiento preciso y exhaustivo de cómo el cerebro y los ojos trabajan en combinación para formar el complejo sistema visual humano, como resultado de estas sutiles modificaciones se produce una reducción significativa del tamaño de los datos necesarios para representar la imagen, con la particularidad de que los efectos en la calidad visual son prácticamente nulos, la posibilidad de que esas modificaciones sean apreciables por el ojo humano depende típicamente del grado de compresión que se utilice y de la técnica usada.

Hoy en día la compresión de imágenes es crucial, el crecimiento de la informática multimedia (las computadoras se utilizan para la videoproducción, difusión,etc) También es muy relevante el papel que se desempeña en temas como la videoconferencia, imágenes médicas, envío de FAX, el control remoto de aplicaciones militares, etc.

- El estándar JPEG (*Joint Photographic Experts Group*).

Este estándar desarrollado en forma conjunta por los comités de CCITT y la International Standards Organizaciotn (ISO), además de más de una docena de empresas como IBM,NEC,Digitl,Kodak y Polaroid, es un sistema abierto y simétrico de compresión lossy. Casi siempre implementado con una simple especificación llamada FIF(JPEG File Interchage Format),que soporta imágenes de 8-bits (escala de grises)y 24-bits (RGB),sin embargo algunas organizaciones han creado versiones personalizadas de este estándar.

La compresión de imágenes JPEG contiene una serie de técnicas avanzadas, la principal, la que hace la compresión real de la imagen es la denominada Discrete Cosine Transform(DCT) seguida por una cuantificación que elimina la información redundante (las partes “invisibles”) el objetivo que persigue este estándar es la compresión de imágenes fijas con un gran margen de calidad y niveles de compresión altos, Presentando las siguientes características:

- Resolución arbitraria de la imagen original
- Varios modelos de color
- Varios formatos de muestreo
- Algoritmos de compresión con pérdidas y sin pérdidas
- La compresión depende del contenido de la imagen y de la calidad deseada

En términos generales el estándar JPEG funciona así:

Primero convierte los colores en sistema LAB, después se deshace de la mitad o tres cuartas partes de la información de color (dependiendo de la implementación). En seguida aplica un logaritmo llamado DCT (Discrete Cosine Transform) que

analiza bloques de 8 en 8 pixeles del archivo. Por cada bloque genera una serie de números que representan en un espectro de lo más general a lo más fino. Los primeros números representan el color general del bloque, mientras que los últimos representan los detalles más finos. Este sistema se basa en la percepción humana, la información más general es más importante.

Dependiendo del grado de compresión que se le pida varía la cantidad de estos números DCT entre menos compresión más números y más detalle, entre mayor compresión menos números y menos detalles. En un último paso JPEG utiliza la compresión Huffman para guardar más eficientemente la información.

Para obtener los mejores resultados del estándar JPEG es importante utilizarlo en el tipo de imágenes para las que fue desarrollado, imágenes naturales (fotografías) en las cuales hay degradados y transiciones de color irregulares, utilizar varias veces la compresión JPEG causa que la calidad de la imagen se pierda, por lo que se recomienda utilizarla en la última etapa únicamente, cabe destacar que JPEG es actualmente el formato más utilizado para almacenar y transmitir archivos de fotos en la Internet.

Se presentara en detalle cada uno de los pasos necesarios para llevar acabo la compresión de una imagen mediante el estándar JPEG. Tales pasos son los siguientes:

Paso 1: Descomposición en bloques o subimágenes. Usando la luminancia (brillo) y la crominancia (color) se consigue mejor compresión, estos valores son otra forma de expresar los valores RGB de una imagen, el ojo es mucho más sensible a la señal de luminancia que a las señales de crominancia, así que no hace falta tanta precisión en estas últimas. Primero se computa la luminancia, Y, y las dos crominancias, I y Q (para NTSC), de acuerdo a las siguientes fórmulas:

$$Y = 0.3R + 0.59G + 0.11B$$

$$I = 0.60R - 0.28G - 0.32B$$

$$Q = 0.21R – 0.52G + 0.31B$$

Para PAL, las crominancias se llaman U y V y los coeficientes son diferentes, pero la idea es la misma. SECAM es diferente a NTSC y PAL.

Una vez obtenidas las matrices Y, I y Q, se hace la media de bloques cuadrados de cuatro píxeles en las matrices I y Q para reducir cada una de las matrices de crominancia a un cuarto de su dimensión original, con lo cual se obtiene un factor de compresión de 2:1, para finalizar este proceso de descomposición en bloques, se divide cada matriz en bloques de 8x8 pixeles, cada bloque contiene 64 números que tienen valores que oscilan entre 0 y 255.

Paso 2: cálculo de la DCT. Se aplica la DCT a cada uno de los bloques Y, I y Q, generando para cada uno de ellos una nueva matriz de 8x8 compuesta por los coeficientes de las componentes de frecuencias espaciales. El valor de estos coeficientes disminuye rápidamente cuando se van alejando del origen de la matriz, terminando generalmente en una serie de ceros. En teoría un DCT es sin pérdidas, pero en la práctica, usando números en coma flotante siempre se introduce algo de error de redondeo que resulta en una pérdida de información.

Paso 3: Discriminación por umbral y cuantificación. Esta etapa tiene en cuenta las particularidades de la visión humana: consiste en poner a cero los coeficientes inferiores a un valor predeterminado y en codificar los coeficientes restantes con una precisión decreciente a medida que la frecuencia aumenta. Esta transformación se hace dividiendo cada coeficiente en la matriz DCT 8x8 por su peso correspondiente en una tabla llamada tabla de cuantización. Los valores de la tabla de cuantización no son parte del estándar JPEG. Cada aplicación debe tener su tabla, permitiendo controlar la pérdida que se presenta en el proceso de compresión. Si todos los pesos en la tabla son 1, la transformación no hace nada. Pero si los pesos se incrementan rápidamente desde el origen, las frecuencias espaciales más altas son prácticamente anuladas y desechadas.

En este paso se disminuye la redundancia psicovisual en las imágenes, JPEG

incorpora las características del sistema visual humano en el proceso de compresión a través de la especificación de matrices de cuantificación, se conoce que la respuesta en frecuencia del sistema visual humano decae con el incremento de la frecuencia espacial. Además, este decaimiento es más rápido en los dos canales de crominancia, esto implica que una pequeña variación en la intensidad es más visible en regiones de variación lenta que en las regiones de variación rápida, y también más visible en la luminancia comparada con una variación similar en la crominancia.

Paso 4: Barrido en Zig-Zag. Con la excepción del coeficiente DC que se trata por separado, los 63 coeficientes AC se leen en zig-zag para transformar la matriz en una cadena de datos en serie. Recorrer el bloque de izquierda a derecha y de arriba abajo no concentrará los ceros juntos, así que se usa un patrón de recorrido en zig-zag, con esto se preparan los datos correspondientes a cada bloque para el siguiente paso dentro del proceso.

Paso 5: codificación Entrópica de Huffman (VLC). Esta última etapa consiste en codificar los coeficientes con una longitud tanto más corta cuanto más frecuentes sean estadísticamente.

Conclusiones y recomendaciones: El JPEG puede parecer complicado, pero es porque es complicado, aunque como ofrece una compresión de 20:1 o mejor, es ampliamente usado, decodificar una imagen JPEG requiere ejecutar el algoritmo al revés. A diferencia de otros algoritmos vistos, JPEG es muy simétrico: decodificar lleva tanto tiempo como codificar.

Es bastante interesante que debido a las propiedades matemáticas de la DCT, es posible llevar a cabo ciertas transformaciones geométricas (p.e. rotación de imagen) directamente en la matriz transformada, sin regenerar la imagen original.

El sucesor de este estándar es el JPEG2000 cuya principal diferencia con el actual es que en vez de realizar la transformación DCT, JPEG 2000 utiliza la transformación Wavelet, la ventaja de JPEG 2000 es que los bloques de JPEG se

eliminan y se remplazan con una imagen generalmente más difusa, esta difusión de JPEG 2000 es preferible frente a los bloques de JPEG, en general es un asunto de preferencia personal, en cualquier caso el ratio de compresión de JPEG 2000 es más alto. Para ratios de compresión moderados, JPEG 2000 produce imágenes típicamente un 25% inferiores en tamaño de archivo que JPEG con igual calidad de imagen, el precio a pagar es trabajar con una técnica de compresión mucho más compleja, el estándar también está pensado para aplicarse a imágenes volumétricas y en movimiento, cosa que permitirá que se puedan desarrollar cámaras fotográficas digitales con mayor capacidad y mejores prestaciones.

3.3.3 Compresión de audio

La compresión de audio es un proceso destructivo, los algoritmos empleados para llevar acabo la compresión se aprovechan de las propiedades que presentan las fuentes de sonido y las limitaciones físicas del oído humano.

El sonido es una onda continua que se propaga a través del aire u otros medios, formada por diferencias de presión, de forma que puede detectarse por la medida del nivel de presión en un punto, las ondas sonoras poseen las características propias de las ondas en general, tales como reflexión, refracción y difracción, estas ondas sonoras son digitalizadas por sistemas electrónico-mecánicos para facilitar su almacenamiento y posterior reproducción.

El almacenamiento y reproducción del sonido se hace siguiendo dos criterios:

- Almacenar la forma de onda de la forma más fiel posible para reproducirla después.
- Establece las reglas por las que se formará el sonido, después se aplican estas reglas a las fuentes de sonido para conseguir el sonido deseado.

En el proceso de digitalización del sonido se hace uso de los dos criterios anteriormente mencionados para representar las ondas de sonido como una serie de números discretos.

Digitalización del sonido: La digitalización del sonido mediante el sistema de muestreo y cuantización consiste básicamente en tomar muestras periódicas de la amplitud de la señal, una vez tomada la muestra esta es cuantizada, el proceso de cuantización consiste en hacer corresponder un número definido en un intervalo determinado a una muestra dada.

El teorema de Nyquist garantiza que la frecuencia necesaria para muestrear una señal debe ser el doble de la frecuencia máxima que presenta la señal, así, si las componentes de máxima frecuencia presentan una frecuencia f, la frecuencia de muestreo debe ser como mínimo 2f, como el rango superior de la audición humana está en torno a los 20 Khz, del teorema de Nyquist, la frecuencia que garantiza un muestreo adecuado para cualquier sonido audible será de unos 40 Khz, con una frecuencia de muestreo mayor o iguales a 40 Khz se garantiza que la perdida en la calidad del audio es prácticamente imperceptible, pues con esta frecuencia de muestreo se garantiza que la señal reconstruida a partir de estas muestras es prácticamente idéntica a la original.

En la práctica para obtener sonido de alta calidad se utilizan frecuencias de 44'1Khz (CD), y hasta 48Khz (DAT) otros valores típicos son submúltiplos de la primera, 22 y 11 Khz. Según la naturaleza de la aplicación la frecuencia de muestreo puede disminuir mucho más, con la consecuente pérdida de calidad que corresponde a esta reducción.

Durante el proceso de cuantización a cada muestra se le asigna un número que se determina generalmente dependiendo del valor de la amplitud de la onda en el instante en el que se realizó la toma de la muestra, el numero asignado está definido en un intervalo predefinido y su precisión depende del número de bits con los cuales se almacena el valor cada muestra cuantizada, dicho intervalo tiene una correspondencia directa con la división del eje de amplitud, se puede realizar a intervalos iguales o según una determinada función de densidad, buscando más resolución en ciertos tramos si la señal que se trata tiene más componentes en cierta zona de intensidad, cuantos más bits se utilicen para la división del eje de la

amplitud, mayor será la resolución y por tanto menor el error al atribuir una amplitud concreta al sonido en cada instante. Por ejemplo, 8 bits ofrecen 256 niveles de cuantización y con 16 bits se tienen 65536.

El proceso completo de muestreo y cuantización se denomina PCM (Pulse Code Modulation) PCM se puede interpretar como un método de conversión Análogo/Digital combinado con un almacenamiento digital no comprimido para señales analógicas.

Codificación y compresión: Las necesidades de transmisión y almacenamiento de sonido demandan gran cantidad de recursos, tanto en ancho de banda como en espacio de almacenamiento, para con seguir una reducción en el consumo de estos recursos se recurre a comprimir la información resultante de su digitalización, La percepción auditiva del ser humano no es sistema perfecto de hecho tiene muchas limitaciones, debido a estas limitaciones una cantidad significativa de la información que proporciona el PCM puede desecharse y por consiguiente se va a obtener una reducción del volumen de datos necesarios para representar el sonido..

La digitalización de la señal mediante PCM es la forma más simple de codificación de la señal, y es la que utilizan tanto los CD como los sistemas DAT, como toda digitalización, añade ruido a la señal, generalmente indeseable. Como hemos visto, cuantos menos bits se utilicen en el muestreo y la cuantización, mayor será el error al aceptar valores discretos para la señal continua, esto es, mayor será el ruido, para evitar que el ruido alcance un nivel excesivo hay que emplear un gran número de bits, de forma que a 44'1 Khz. y utilizando 16 bits para cuantizar la señal, uno de los dos canales de un CD produce más de 700 kilobits por segundo (kbps) como veremos, gran parte de esta información es innecesaria y ocupa un ancho de banda que podría liberarse, a costa de aumentar la complejidad del sistema decodificador e incurrir en cierta pérdida de calidad. De esta forma, el compromiso entre ancho de banda, complejidad y calidad es el que produce los diferentes estándares del mercado.

Un modo mejor de codificar la señal es mediante PCM no-lineal o cuantización

logarítmica, que como ya comentamos consiste en dividir el eje de la amplitud de tal forma que los escalones sea mayor cuanta más energía tiene la señal, con lo que se consigue una relación señal/ruido igual o mejor con menos bits. Con este método se puede reducir el canal de CD audio a 350 kbps, lo cual evidentemente es una mejora sustancial, aunque puede reducirse mucho más, otros sistemas similares nos llevan a la cuantización adaptativa (APCM), diferencial (DPCM) y la mezcla de ambas, ADPCM. Así prosigue la reducción del ancho de banda, pero sin llegar a los niveles que proporciona el tener en cuenta los efectos del enmascaramiento.

Codificación sub-banda (SBC). La codificación sub-banda o SBC (sub-band coding) es un método potente y flexible para codificar señales de audio eficientemente, a diferencia de los métodos específicos para ciertas fuentes, el SBC puede codificar cualquier señal de audio sin importar su origen, ya sea voz, música o sonido de tipos variados, el estándar MPEG Audio es el ejemplo más popular de SBC.

El principio básico del SBC es la limitación del ancho de banda por descarte de información en frecuencias enmascaradas, el resultado simplemente no es el mismo que el original, pero si el proceso se realiza correctamente, el oído humano no percibe la diferencia.

La mayoría de los codificadores SBC utilizan el mismo esquema.

Paso 1: Un filtro o un banco de ellos, o algún otro mecanismo descompone la señal de entrada en varias subbandas.

Paso 2: Se aplica un modelo psicoacústico que analiza tanto las bandas como la señal y determina los niveles de enmascaramiento utilizando los datos psicoacústicos de que dispone.

Paso 3: considerando estos niveles de enmascaramiento se cuantizan y codifican las muestras de cada banda: si en una frecuencia dentro de la banda hay una

componente por debajo de dicho nivel, se desecha. Si lo supera, se calculan los bits necesarios para cuantizarla y se codifica.

Paso 4: se agrupan los datos según el estándar correspondiente que estén utilizando codificador y decodificador.

La decodificación es mucho más sencilla, ya que no hay que aplicar ningún modelo psicoacústico. Simplemente se analizan los datos y se recomponen las bandas y sus muestras correspondientes.

3.3.3 Compresión de video

El vídeo es solo una reproducción en forma secuencial de imágenes, que al ser proyectadas con una determinada velocidad y continuidad dan la sensación al ojo humano de apreciar el movimiento continuo, los componentes del video son básicamente imágenes y sonido.

La creciente potencia de los microprocesadores, las grandes capacidades de los discos duros y su alta velocidad de transferencia, el abaratamiento del acceso a Internet y la aparición de algoritmos de compresión realmente eficientes están haciendo posible la manipulación y el tráfico de video entre las computadoras personales de millones de personas alrededor del mundo.

En la actualidad la transmisión de vídeo sobre redes de telecomunicaciones está llegando al punto de convertirse en un sistema habitual de comunicación, hoy en día se está utilizando el video para ver películas, comunicarnos con otras personas, para hacer diagnósticos en medicina, videoconferencia, distribución de TV, vídeo bajo demanda y distribución de multimedia en Internet, entre otras muchas aplicaciones.

La compresión de video surge de la necesidad de transmitir imágenes a través de un canal con limitaciones de ancho de banda, los métodos de compresión de video recurren a los procedimientos generales de compresión de datos; pero además

aprovechan la redundancia espacial de cada uno de los fotogramas, la correlación entre puntos cercanos y la menor sensibilidad del ojo a los detalles finos de las imágenes fijas, para imágenes animadas, se saca provecho de la redundancia temporal entre imágenes sucesivas.

La compresión de vídeo la es posible hacer de dos maneras:
Compresión espacial o Intra-Frame: se aprovecha la redundancia de información que hay en la imagen de cada fotograma, es compresión consiste en aplicar un algoritmo de compresión de imágenes a cada uno de los fotogramas del video, el algoritmo más usado para comprimir los fotogramas es el estándar JPEG.

Compresión temporal o Inter-Frame: se aprovecha la redundancia de información que hay entre fotogramas sucesivos.

Compresión Espacial O Intra-Frame. La Compresión Espacial o Inter-Frame consiste en la aplicación de un algoritmo de compresión de imágenes a cada uno de los fotogramas que componen el video, siendo el estándar JPEG el más usado, el tema de compresión de imágenes fija fue tratado en el apartado de Compresión de Imágenes.

Compresión Temporal O Inter-Frame. La redundancia temporal es reducida primero usando similitudes entre sucesivas imágenes, haciendo uso de información de imágenes procesadas con anterioridad, cuando se usa esta técnica, sólo es necesario enviar la diferencia entre las imágenes, es decir las zonas de la imagen que han variado entre dos fotogramas consecutivos, lo que elimina la necesidad de transmitir la imagen completa.

Los estándares MPEG. En el año de 1990, la ISO promueve la creación del MPEG (Moving Pictures Expert Group), conformada por expertos procedentes de áreas como las telecomunicaciones, informática, electrónica, radio difusión, etc. El primer trabajo de este grupo se conoció como la norma ISO/IEC 11172, mucho más conocida como MPEG-1, en el año 1992, la idea inicial era la de permitir el

almacenamiento y reproducción en soporte CD-ROM con un flujo de transmisión de datos del orden de 1,5 Mbits/s, transportando tanto imagen como sonido.

Debido a que la calidad en la compresión de video en el estándar MPEG-1 era de baja calidad y no era apto para algunas aplicaciones, se creó la norma ISO/IEC 13818, mucho más conocida con el nombre de MPEG-2. Esta norma permite un flujo de transmisión hasta el orden de los 20 Mbits/s, transportando tanto imagen como sonido.

Posteriormente se crearía MPEG-4 que está encaminada a la transmisión de datos del orden de los 8 a 32 Kbits/s, norma que será utilizada en las aplicaciones de video conferencia o video teléfono.

Ahora que ya hemos conocido un poco sobre los orígenes de los estándares MPEG, se presentara como es su funcionamiento.

Empezaremos con la compresión de audio. La compresión de audio MPEG se hace muestreando la onda a 32 kHz, 44.1 kHz ó 48 kHz. Puede manejar mono, estéreo disjunto (cada canal comprimido separadamente), o estéreo junto (se explota la redundancia intercanal). Se organiza como tres capas, cada una de ellas aplicando optimizaciones adicionales para conseguir más compresión (y a mayor costo), la capa 1 es el esquema básico, esta capa se usa, por ejemplo, en el sistema de cinta digital DCC. La capa 2 añade al esquema básico localización de bits adicional, se usa para audio en CD-ROM y bandas sonoras de películas.

La capa 3 añade filtros híbridos, cuantización no uniforme, codificación Huffman y otras técnicas avanzadas.

El audio MPEG puede comprimir un CD de música en 96 kbps sin pérdida de calidad perceptible, para el oído humano. Hay situaciones en las cuales el bitrate debe aumentar, por ejemplo para comprimir un concierto de piano, se necesitan al menos 128 kbps. Esta diferencia es porque la relación señal/ruido de los distintos géneros de música y en general todos los sonido difieren de uno a otro.

La compresión de audio se lleva a cabo haciendo una transformada rápida de Fourier en la señal de audio para transformarla del dominio del tiempo al dominio de la frecuencia, el espectro resultante se divide en 32 bandas de frecuencia, y cada una es procesada de forma separada, cuando hay presentes dos canales estéreo, la redundancia inherente de tener dos fuentes de audio altamente superpuestas también se explota, para ver más acerca de los algoritmos de compresión de audio, ver el apartado *Compresión de Audio* incluido en este capítulo.

Después de entender como es procesado el audio mediante MPEG, continuemos con el objetivo principal de MPEG, la compresión de una secuencia de fotogramas.

La forma más primitiva de codificación de video solo explota la redundancia espacial, codificando cada fotograma separadamente con JPEG, a esta forma de codificación de video se le conoce como MJPEG, esta aproximación se usa a veces, especialmente cuando se necesita acceso aleatorio a cada fotograma, como en edición de producciones de vídeo. De este modo, es alcanzable un ancho de banda comprimido de 8 a 10 Mbps.

Para escenas donde la cámara y el fondo son estacionarios y uno o dos actores se mueven lentamente, casi todos los píxeles serán idénticos de fotograma a fotograma, aquí, restar cada fotograma al anterior y aplicar JPEG a la diferencia estaría bien, hay escenas donde la cámara hace zoom o se mueve, en estos casos la compresión mediante MJPEG falla estrepitosamente, pues los fotogramas consecutivos cambian en gran medida con respecto fotogramas contiguos. Para este caso lo que se necesita es alguna forma de compensar este movimiento y es allí donde MPEG entra a suplir esta necesidad existente.
Para iniciar con él la explicación de MPEG, se va a empezar por el final del proceso. Es decir, con la salida que se genera una vez aplicado el algoritmo a una secuencia de fotogramas. La salida MPEG consiste en cuatro tipos de fotogramas:

a) Intra (I): autocontenidos, solo compresión espacial (como JPEG)

b) Previstas (P): referidos al fotograma P o I anterior. Compresión temporal por macrobloques. Un macrobloque puede ser: inalterado: no modificado respecto al fotograma de referencia, desplazado y se describe por un vector de movimiento y eventualmente una corrección(diferencia respecto al original) o nuevo: se describe por compresión espacial (como un fotograma I)

c) Bidireccionales (B): compresión temporal con interpolación; referido al fotograma P o I anterior y al fotograma P o I posterior. Se obtiene máxima compresión a cambio de una alta complejidad. Suavizan la imagen y reducen el ruido.

d) DC-coded (D): Medias de bloques usadas para avance rápido.

Cabe aclarar a que no todas las variantes de MPEG soportan todos los tipos de fotogramas presentados anteriormente.

Una secuencia de fotogramas MPEG está constituida por un fotograma I y los siguientes fotogramas son P o B hasta el comienzo del otro fotograma I, a esta secuencia base se le denomina GOP (Group Of Pictures) para factores de compresión altos se utiliza un número grande de fotogramas P o B, haciendo que las GOPs aumenten de tamaño considerablemente; sin embargo un GOP grande evita recuperar eficazmente una transmisión que ha llegado con errores, esto hace necesario que se introduzcan fotogramas I periódicamente.

La aparición de fotogramas I periódicamente se hace por tres razones:
Los estándares MPEG pueden ser usados para una transmisión multicast, con espectadores conectándose cuando quieran, si todos los fotogramas dependiesen de sus predecesores hasta el primer fotograma, cualquiera que perdiera el primer fotograma nunca podría decodificar los fotogramas subsecuentes.

Si se recibiera un fotograma con error, no se podría seguir decodificando.

Sin los fotogramas I, mientras se hace un avance o retroceso rápido, el decodificador tendría que calcular cada fotograma sobre el que se ha pasado para saber el valor íntegro del fotograma sobre el cual se ha parado.

Ya sabemos que los fotogramas I se codifican haciendo uso de una técnica de compresión de imágenes, ahora se presentara como se codifican los fotogramas P, B y D.

Los fotogramas P codifican diferencias entre fotogramas. Se basan en la idea de los macrobloques, un macrobloque se codifica buscando su fotograma previo o algo sólo ligeramente diferente de ello, para ilustrar una situación en la cual es adecuado el uso de fotogramas P, imaginemos dos fotogramas consecutivos que tienen el mismo fondo, pero que difieren en la posición de un objeto, los macrobloques que contienen la escena de fondo concordarán exactamente, pero los macrobloques que contienen el objeto tendrán una diferencia de posición y habrá que darles un seguimiento.

Los estándares MPEG no especifican cómo buscar los macrobloques o cómo de buena debe ser una coincidencia entre estos, esto se deja para cada implementación, por ejemplo, una implementación puede buscar un macrobloque en la posición actual en el fotograma anterior y en todas las demás posiciones con un desplazamiento X en la dirección *x* y un desplazamiento Y en la dirección *y*, para cada posición, el número de coincidencias en la matriz de luminancias podría ser computado, la posición con la más alta puntuación sería declarada ganadora, habiendo definido previamente un umbral, de otro modo, se diría que el macrobloque se ha perdido, el anterior es solo un ejemplo de cómo podría llevarse acabo la búsqueda de los macrobloques, es de esperarse que las implementaciones reales sean más complejas y sofisticadas.

Si se encuentra un macrobloque, se codifica cogiendo la diferencia con su valor en

el fotograma anterior (para la luminancia y las dos crominancias) estas matrices de diferencias son entonces objeto de una DCT, cuantización, codificación RLE y codificación Huffman, como en JPEG ,el valor del macrobloque en la salida es el vector de movimiento (cuánto se ha movido el macrobloque en cada dirección), seguido por la lista de números codificada con Huffman. Si el macrobloque no se localiza en el fotograma anterior, el valor actual se codifica con JPEG, como en un fotograma I.

Claramente la aplicación del algoritmo es altamente asimétrico, una implementación es libre para probar cada posición plausible en el fotograma anterior si quiere, en un intento de localizar cada último macrobloque. esta aproximación minimizará la salida MPEG a expensas de una codificación muy lenta, podría ser buena para la codificación una única vez de una biblioteca de películas pero no sería aplicable para videoconferencia en tiempo real.

Similarmente, cada implementación es libre para decidir lo que constituye un macrobloque "encontrado". Esta libertad permite a los implementadores competir en la calidad y velocidad de sus algoritmos, pero produciendo siempre MPEG estándar. No importa qué algoritmo de búsqueda se use, la salida final es la codificación JPEG del macrobloque actual o la codificación JPEG de la diferencia entre el macrobloque actual y uno en el fotograma anterior en un desplazamiento especificado desde el actual.

Los fotogramas B son similares a los fotogramas P, excepto que permiten que el macrobloque de referencia esté en un fotograma previo o en uno posterior, esta libertad adicional permite compensación de movimiento adicional, para codificar fotogramas B, el codificador necesita tener tres fotogramas decodificados en memoria a la vez: el anterior, el actual y el siguiente., aunque los fotogramas B dan la mejor compresión.
Conclusiones y recomendaciones: Todos los esquemas de codificación que se han discutido están basados en el modelo de codificación con pérdidas seguida de transmisión sin pérdidas ni el JPEG ni el MPEG, por ejemplo, pueden recuperarse

bien de la pérdida o daño de paquetes, una aproximación diferente a la transmisión de imágenes es transformar las imágenes de forma que se separe la información importante de la menos importante (como hace DCT, por ejemplo) entonces añadir una considerable cantidad de redundancia (incluso duplicar paquetes) a la información importante y nada a la menos importante, si algunos paquetes se pierden o dañan, puede ser todavía posible mostrar imágenes razonables sin retransmisión, estas ideas son especialmente aplicables a transmisiones multicast, donde el feedback desde cada receptor es imposible de cualquier modo.

3.3.4 Compresión de video moderna

En este capítulo hablaremos acerca de algunas tecnológicas que se encuentran en el mercado, donde nos proponen mejorar en la calidad de la compresión de nuestros videos, todo empezó alrededor de 2004 cuando la ITU-T video coding experts group, donde los estudios de avances tecnológicos permitieron la creación de una nueva forma de compresión de video, esto lo hicieron con base la norma anterior H.264,los cuales hicieron mejores sustanciales orientadas a una mejor compresión, tiempo después de hacer las suficiente pruebas de la nueva norma H.265, se sorprendieron por los cambios que esta norma nos pudiera traer, donde poco a poco empezó a ganar terreno en el campo de la compresión de video, donde se promete revolucionar el *streaming* de vídeo así como las transmisiones y descargas de vídeo en dispositivos móviles, el motivo por el cual este fue muy aceptado es que reduce a la mitad el ancho de banda necesario para la transmisión de vídeo y todo ello sin la temida pérdida de calidad que sí que encontramos en otras alternativas del mercado.

Estamos hablando de un método de codificación de vídeo de alta eficiencia (High Efficiency Video Coding o HEVC) el resultado, la posibilidad de ver en *streaming* contenidos audiovisuales con resoluciones que pueden incluso superar el Full HD o 1080p y con un peso mucho más reducido.

El gran motivo de su éxito es que con la misma compresión del video de la norma H.264 AVC logra con la misma calidad de la compresión de video se logra que este a la mitad de la tasa de bits, logrando una importante reducción en el tamaño de los vídeos.

Un ejemplo donde se logra dimensionar la gran importancia de esta nueva norma es que para un video Full HD reducido con H.264 tendría un peso de 2.5 Gb ahora un video Full HD reducido con H.265 pesaría 1Gb, una gran diferencia entro estos dos sin embargo este no ofrece una perceptible pérdida de calidad.

De esta forma los grandes beneficiados para usar este tipo de códec serían los dispositivos móviles (teléfonos o tabletas) que verían más optimizada la emisión de vídeo mediante navegadores web en HTML5, además y junto con la resolución 1080p, este códec también beneficia al vídeo en resolución 4K para usar en la nueva generación de televisores.

Codificar con H.265 aumenta considerablemente el tiempo requerido, no obstante, la mejora que se consigue, teniendo en cuenta el gran ahorro de tamaño del archivo en relación con la poca diferencia de calidad o pérdida de detalles, hacen de este nuevo códec una opción muy interesante para el vídeo por internet, además, aunque no esté muy extendido su uso, parece ser que la mayoría de reproductores ya soportan su lectura.

4. CONCLUSIONES Y RECOMENDACIONES

- El conocimiento claro de los conceptos que involucra la teoría de la información, se constituye en una valiosa herramienta en el momento de aventurarse a explorar cualquier técnica de compresión de datos.
- Una visión estadístico-matemática de los datos facilita la conceptualización e inventiva de nuevos algoritmos de compresión, pues de alguna forma libera al investigador de las particularidades que suponen las estructuras de datos reales y sus restricciones.
- Es claro que los medios de almacenamiento actuales son más que suficientes para la mayoría de los usuarios hogareños de computadoras personales y el tamaño de los discos duros no es algo de lo que estos deban preocuparse, además, el uso de medios de almacenamiento ópticos para respaldar los datos es un gran alivio a la hora de mantener bajos los volúmenes de datos en disco duro.
- En ambientes corporativos un buen manejo del ciclo de vida de la información,

asegura un uso óptimo de los medios de almacenamiento, aunque en estos ambientes los altos volúmenes de datos son algo cotidiano, es válido pensar que complejos sistemas de información distribuidos serán los que predominen en el futuro facilitando y aligerando la carga se pudiera requerir a los dispositivos de almacenamiento.

- Los medios de almacenamiento son cada día más robustos en capacidad y seguridad. Transportar 9 Gb de datos en un DVD en uno de nuestros bolsillos no es nada sorprendente; pero siempre se va a desear llevar datos de un lado a otro y los medios de almacenamiento extraíbles todavía no son lo suficientemente robustos o en su defecto los suficientemente baratos. Aun debemos comprimir el video.
- El ancho de banda presente en las redes LAN de hoy se considera suficiente para la mayoría de las aplicaciones. No es allí donde la compresión juega un papel relevante; es en los enlaces WAN donde se presentan picos de botella y es allí donde el transporte de datos comprimidos marca una diferencia en los tiempos que requieren las transferencias. El transporte de datos comprimidos en las redes de computadoras estará allí siempre y cuando represente un ahorro de recursos.
- El uso de modelos estadísticos en la concepción de nuevos algoritmos de compresión esquematizan a dichos algoritmos en unas tazas de compresión muy similares a los existentes y en últimas terminan siendo irrelevantes, pues generalmente son algoritmos de alta complejidad computacional.
- En el campo de la compresión basada en diccionario, es claro que el concepto de diccionario no va a variar mucho en el futuro. Las mejoras en este tipo de compresores vendrán de las implementaciones adaptativas que estarán enfocadas en eliminar por completo la necesidad de entregar un esquema de diccionario proveniente del proceso compresor al descompresor.
- Una gran oportunidad resultan ser los algoritmos basados en transformadas, especialmente aquellas transformadas sin perdida, en la mente de los investigadores, las interminables cadenas de bits son rotadas, dobladas, enrolladas, giradas, truncadas, concatenadas y cualquier otra transformación que por alguna mente inquieta se pueda cruzar, en este campo no hay

paradigmas arraigados y esto constituye un atractivo más para los investigadores, pensar en algoritmos de compresión adaptativos basados en transformadas sin perdida es pensar en lo que será el futuro de la compresión sin perdida.

- La compresión de multimedia no es de esperar que cambie mucho en su forma general, fácilmente podría surgir en este campo transformadas matemáticas que generen resultados más óptimos que las actuales en lo referente a calidad; pero las mejoras en las tasas de compresión no provendrán del uso de estas nuevas transformadas, es de esperarse que las mejoras en las tasas de compresión de material multimedia provengan de la sofisticación de los algoritmos de búsqueda encargados de hallar las redundancias presentes en dicho material.

BIBLIOGRAFÍA

JESÚS SILVA, El almacenamiento de información en las computadoras, Agosto 27 de 2001, Copyright 2000 CLAVE EMPRESARIAL COM, S.A. DE C.V. <http://www.claveempresarial.com/principiantes/notas/nota010827b.shtml>

JOSE JAIME MESEGUER NAVARRO, DIEGO SEVILLA RUIZ, Compresión de Datos en las Comunicaciones, Junio de 1997, España.jmeseguer@hotmail.com, dsevi@mistal.dif.um.es.<http://intelec.dif.um.es/~aike/tic/compresion/Compress.htm>

JUAN ALBERTO SIGÜENZA PIZARRO, Introducción general a la compresión de

datos multimedia Necesidad de la compresión. Escuela Técnica Superior de Informática Universidad Autónoma de Madrid. jalberto.siguenza@ii.uam.es. <www.ii.uam.es/~siguenza/Introcompr.ppt>

JAMES D. MURRAY, WILLIAM VANRYPER. Encyclopedia of Graphics File Format, Copyright 1994 O'Reilly & Associates, Inc. U.S.A., editor Deborah Russell, Julio de 1994.

INSTITUTO COLOMBIANO DE NORMAS TÉCNICAS. Normas Colombianas para la presentación de trabajos de investigación. Segunda actualización. Santa fe de Bogotá D.C.:ICONTEC, 1996.

TIM BELL, MATT POWELL, JOFFRE HORLOR, ROSS ARNOLD. Canterbury Corpus. Canterbury University. Nueva Zelanda. Noviembre 20, 2001. http://corpus.canterbury.ac.nz/index.html

IVAN DARIO MESA I.E.E. EE.PP.M Teoria de la Informacion de Shannon. Artículo de Revista Título de la publicación OMEGA No. 4 [1996]. Medellin Facultad de Ingenieria Electrica y Electronica U.P.B, 1991-1996

K. Sayood, Introduction to Data Compression, 2nd Ed., MK Publishers, 2000.

M. Bosi and R. Goldberg, Introduction to Digital Audio Coding and Standards, Kluwer, 2003

Avid Roman-Gonzalez. Clasificacion de Datos Basado en Compresion. Revista ECIPeru, 2012, 9 (1), pp.69-74.

Grupo de Quimiometría y Cualimetría de Tarragona. Técnicas de Laboratorio, 272 (2002) 412-416.

Richardson I. E.G. (2003). H.264 and MPEG–4 Video Compression:

Video Coding for Next–generation Multimedia, Wiley

Wenger A. (2003). H.264/AVC Over IP. IEEE Transactions on Circuits and

Systems for Video Technology, Vol. 13, pp. 645–656, julio.

Huang Y.W. (2005). Analysis, Fast Algorithm, and VLSI Architecture

Design for H.264/AVC Intra Frame Coding. IEEE Transactions on Circuits and Systems for Video Technology, Vol. 15, pp. 378–401, Marzo.

Flierl M. and Girod B. (2003). Generalized B Picture and the Draft H.264/AVC

Video–Compression Standard. IEEE Transactions on Circuits and Systems

for Video Technology, Vol. 13, pp. 587–597, Julio.

Colom Palero, RJ. (2001). Estudio e implementación de la transformada Wavelet para la compresión de imágenes y video [Tesis doctoral no publicada]. Universitat Politècnica de València. doi:10.4995/Thesis/10251/4261.

Sayood, K. 1996. Introduction to Data Compression. Morgan Kaufmann Publishers, INC.San Francisco, California, USA.